FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Low Code Solution for IoT Testing

**Hugo Cunha**


Mestrado Integrado em Engenharia Informática e Computação

Supervisor: João Pascoal Faria

Co-Supervisor: Bruno Lima

July 16, 2019

# Low Code Solution for IoT Testing

**Hugo Cunha**

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Doctor Ana Cristina Ramada Paiva
External Examiner: Doctor Miguel António Sousa Abrunhosa Brito
Supervisor: Doctor João Carlos Pascoal Faria
July 11, 2019

# Abstract

Over the last few years, there has been an increase in the use of IoT devices. Nowadays, IoT systems rely on device-to-device communication, based on a centralised server in the cloud. In the near future, it is expected to have around 50 billion devices connected to the internet and to each other. In areas such as e-health and automotive, which are having a great expansion lately, it is important to guarantee a correct integration of these ecosystems because failures from such interconnected network can bring very dangerous outcomes to human lives. In these scenarios, it is not only important to guarantee singular software and hardware testing, but also to consider the systems as a whole, and to test it as such. Also, companies show a relevant concern for the lack of software testing frameworks and also a certain gap between the actual and the desired status of test automation of such systems. However, existing integration testing techniques are not satisfactory and are not properly addressed. From the existing ones, there are a few flaws we can point out: they are very much focused in the platform they were developed in.

With the intent of solving the problems presented before, it was developed a low-code solution based in a visual interface which uses a flow-based approach design so that the user can easily, and graphically, design the integration scenario and furthermore execute the tests and observe the results. Such framework is able to interpret the scenario described graphically, generate a configuration file and deliver it to an integration testing framework so that the tests are executed.

The visual interface was validated by the execution of an usability test with a set of users with different technical knowledge in order to make the process of integration testing available for a larger range of users. The results of the usability test were very positive, having had more than 4 points (out of 5) on every metric evaluated.

# Resumo

Durante os últimos anos, houve um aumento na utilização de dispositivos IoT. Atualmente, os sistemas IoT dependem da comunicação de dispositivo-para-dispositivo, com base num servidor centralizado na *cloud*. Num futuro próximo, espera-se que haja cerca de 50 mil milhões de dispositivos ligados à internet e uns aos outros. Em áreas como e-health e o setor automóvel, que estão a ter uma grande expansão ultimamente, é importante garantir uma integração correta desses ecossistemas, pois falhas de tais redes interconectadas podem trazer resultados indesejados e que colocar vidas humanas em perigo. Neste tipo de cenários é importante garantir tanto testes de software e hardware, mas também considerar os sistemas como um todo e testá-los como tal. Além disso, as empresas têm demostrado uma crescente preocupação com a falta de soluções de teste de software e também com uma certa discrepância entre o estado atual e o estado desejado da automação de teste destes sistemas. No entanto, as técnicas de teste de integração existentes não são satisfatórias e não são adequadamente tratadas. Das existentes, há algumas falhas que podemos apontar, como por exemplo serem muito focadas na plataforma em que foram desenvolvidas.

Com o intuito de resolver os problemas apresentados anteriormente, foi desenvolvida uma solução *low-code* baseada numa interface visual que utiliza um design *flow-based* para que o utilizador possa desenhar graficamente o cenário de integração, executar os testes e observar os resultados. Essa solução é capaz de interpretar o cenário descrito graficamente, gerar um ficheiro de configuração e fornecê-lo a uma *framework* de teste de integração para que os testes sejam executados.

A interface visual foi validada pela execução de um teste de usabilidade com um conjunto de utilizadores com diferentes conhecimentos técnicos, com o objetivo de facilitar o processo de testes de integração para um maior número de utilizadores. Os resultados do teste de usabilidade foram muito positivos, tendo mais de 4 pontos (de 5) em cada métrica avaliada.

# Acknowledgements

In this section I would like to dedicate to the people that, in some way, contributed to the conclusion of my course and dissertation.

I'd like to start by thanking all my family and my friends who helped me throughout the past 5 years of the INFORMATICS ENGINEERING course. I'd like to give a special spotlight to my mother who was an unconditional will power machine during the last 5 years of my informatics engineering course and my entire life. The second spotlight would go to my cousin, who in the first years of the course has helped me tremendously in the introduction to programming, which I had a certain difficulty to familiarize with.

I'd like to thank the invaluable help of my supervisors, Professor João Pascoal Faria, by his comments, points of view and ideas for the development of this thesis and Bruno Lima, co-supervisor, for his comments and suggestions.

Lastly, I'd like to thank the PhD student, João Pedro Dias, working in the software engineering lab who has helped a few times in questions regarding the design of the application or some implementation details.

Hugo Diogo Queirós Cunha

*"You should be glad that bridge fell down.*
*I was planning to build thirteen more to that same design"*

Isambard Kingdom Brunel

# Contents

CONTENTS

# List of Figures

# List of Tables

# Abbreviations

| | |
|---|---|
| ADT | Abstract Data Type |
| ANDF | Architecture-Neutral Distribution Format |
| API | Application Programming Interface |
| CAD | Computer-Aided Design |
| CASE | Computer-Aided Software Engineering |
| CORBA | Common Object Request Broker Architecture |
| UNCOL | UNiversal COmpiler-oriented Language |
| MQTT | Message Queuing Telemetry Transport |
| HTTP | Hypertext Transfer Protocol |
| CoAp | Constrained Application Protocol |
| OCL | Object Constraint Language |
| SWE | Sensor Web Enablement |
| DPWS | Devices Profile for Web Services |
| SUT | System Under Testing |
| NPM | Node Package Manager |
| SDLC | Software Development Life Cycle |

# Chapter 1

# Introduction

## 1.1 Context and Motivation

Over the last few years, there has been a growth in the usage of IoT devices [FP14]. These small devices have been "turning heads" in terms of robustness, price and general usability [FP14]. From simple devices with the intent of measuring the temperature or humidity of a room to more complex ones, capable of turning the TV on or adjusting the air conditioning, these devices are taking a leap forward both in technology and complexity. With the addition of more complexity, there is a rising problem - guarantee the correct integration, communication and functioning when grouped together.

One of the areas in a certain rise, regarding IoT, is eHealth. eHealth is a somewhat recent area that integrates informatics and health in the same domain [FFC+18]. By other words, is the possibility of the creation of new services in the healthcare domain using the internet and informatics combined. It is the possibility of exchanging healthcare information, via the internet, with many health professionals for a better quality of service. Automotive is also another domain to get certain attention [KH16]. Lately, a large number of companies are attempting to create autonomous vehicles. These vehicles are expected to not only detect all kinds of road hazards - pedestrians, road signs, traffic lights - but also be able to communicate with the infrastructure. The latter part is the most concerning one.

From the previously observed domains (eHealth and automotive) it is perceptible that errors derived from these activities may cause serious damage to human lives [BK08]. It is critical that such infrastructures and systems are tested to guarantee its correct functionality.

Another problem that we can point out is the fact that most of the products developed for such scenarios, are developed on different platforms and even in different programming languages [LF16]. Communication between such devices may be very hard to establish because of the previously described aspects.

For these complex scenarios, which involve a large number of devices developed in multiple and different platforms, it is required to have very well established integration tests to not

only guarantee integration between all devices in the network but also to guarantee its expected behaviours.

Integration testing is an important phase of software testing. It is in this crucial phase that the system components are tested as a whole. At this stage, it is not relevant whether the devices perform their job, individually, but instead as a group.

At the moment, there is a vast range of IoT solution for both testing and development [DCPF18]. However, such tools are either somewhat limited in terms of extensibility, only providing support for large-scale systems or require a substantial technical and programming knowledge to operate.

In order to respond to these trends, companies and investigators, are focusing their efforts on the development of new and more efficient solutions. Healthcare is one of the domains that will benefit from the continuous growth of IoT devices. These small, but very useful devices will, not only make it easier for health professionals to perform their work but also patients will benefit from better monitoring and assistance.

In order to develop this dissertation, it will be used a framework developed in a previous dissertation - Izinto as starting point. Apart from Izinto it will be added a new module which is capable of designing visual test for IoT, automatically execute tests and visualize test results, in the end. Such module must be as easy as possible to use so that users with low technical background can use it.

Izinto [PLF18] is a pattern-based, automated software integration testing framework that facilitates the work of an integration tester. It also abstracts configuration of the system's components by using popular communication technologies and protocols. In addition, it enables behaviour testing automatically, by implementing a set of IoT test patterns out-of-the-box.

In this case, the work of a tester can be summed up to the creation and specification of a configuration file.

Izinto has some limitations mainly due to the fact that it requires the tester to have some technical knowledge because the configuration file must be written by the user in JSON and tests executed in a programming environment.

## 1.2 Objectives and Expected Results

With the intent of solving the problems above mentioned, mainly the limitations Izinto presents. The objective of this dissertation is the development of a low-code solution based on a visual interface for IoT integration testing with the following high-level features:

- Visual definition of the test configuration
- Test execution (linking to an existing framework, such as Izinto)
- Visualization of test results
- Test management (importing and exporting test configurations)

In regard to the high-level requirements of the solution, it must be possible for the user to visually create blocks representing sensors, actuators, applications and notification in a user-friendly

way. It is also imperative to be able to create flows, using such blocks' data flow in order to replicate real scenarios. Another aspect to have great relevance is the test pattern suggestion, according to the scenario designed. As stated before, one of the points the solution must be able to perform, is to reduce is the skill required to use. In order to achieve it, a test pattern suggestion algorithm will present the user tests that can be applied to a certain designed scenario. As such, the user does not need to have any knowledge regarding tests and still be able to execute them. The following step is the connection to a testing framework, in this case Izinto, so that the tests can be executed. After test execution, it must be possible for the user to visualize the results, by coloring the blocks according to the outcome of the test. Lastly, it will allow the user to save and import scenarios, for future test execution.

One important aspect of the solution to be developed is to understand the most adequate visual notation for developing a visual interface with focus on testing, especially, integration testing. There are a considerable number of technologies available and an analysis must be done in order to understand which one, or more than one, may lead to a good result.

## 1.3 Document Structure

Besides the introduction chapter, this document is structured as follows:

- In chapter 2, *Background and State of the Art*, it is made a research of the current state-of-the-art for both IoT testing and development tools as well as some background on software and IoT testing and challenges for IoT
- In chapter 3, *Solution Design and Implementation*, it is presented the solution developed, alongside its architecture and general interface functionalities and flow
- In chapter 4, *Validation*, it is presented the validation of the solution developed by conducting a usability test in order to assess the general quality of the work developed
- In chapter 5, *Conclusion and Future Work*, it is presented an overview of the research made, of the current IoT testing and development tools, on the solution developed, general contributions and future work
- Appendix A, it is presented the material used for the usability test, such as the users' handout, the time logging sheet and the questionnaire present to the users
- Appendix **??**, it is presented the scientific publication written within the scope of this work of dissertation

# Chapter 2

# Background and State of the Art

## 2.1 Software Testing Concepts

The Software Development Lifecycle (SDLC) [LLTT12] is the process of building and maintaining software. It starts in the early stages of the development when the gathering of requirements is made and the design is settled, until the last stages where the solution is validated and tested.

Testing [MSB11] is a crucial phase during the SDLC. At this stage, it is ensured that the program, or components, work as they are supposed to. During this stage, it is important to resolve all bugs and errors associated with the software. The correction of such problems later in the project development may lead to higher costs both financially and time. In the next section, it will be made a special focus on the software testing levels and how they can help to create better quality software.

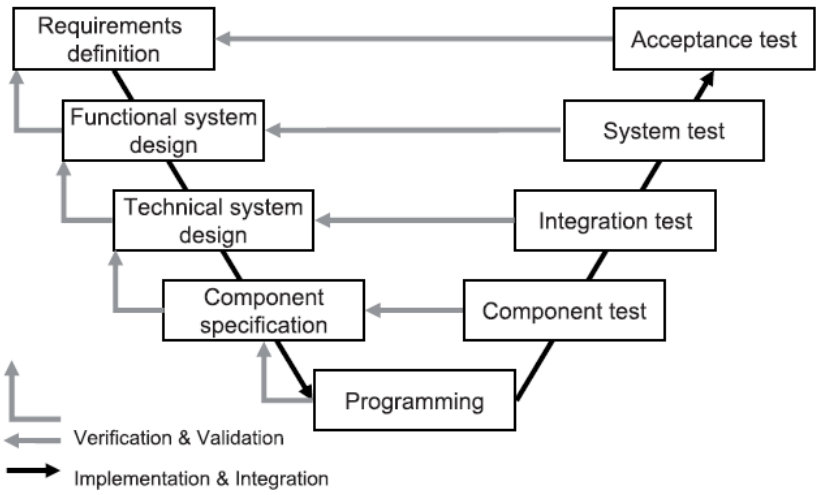



Figure 2.1: The General V-Model
[SLS14]

Spillner *et al.* [SLS14] defined a "V" shaped lifecycle model. In the right-hand side, it is possible to encounter the classic testing levels: components testing, integration testing, system

testing and acceptance testing. In the opposite side, it is presented the refinement of the system or the development process where the system may suffer design changes, until reaching the final stage where it is programmed. The General V-Model is represented in Fig. 2.1. In this dissertation, it will be made a special focus on the right side of the model and, especially, in integration testing.

**Components Testing**, also called Unit Testing, is the process of testing a single component of the software. Unit tests are executed on singular units of source code in order to assess its functionalities. It is an essential process, especially in large software products due to the fact that ensures that the functionalities are implemented and work as intended. Unit testing is also important in test-driven development, where the tests are written prior to the development of source code.

**Integration Testing** is the phase after Unit Testing in which individual software components are combined and tested as a whole. A requisite to run integration tests is that unit tests must have already been executed. Defects from single software components are supposed, and assumed, to have been corrected already. Integration testing is placed just before System Test. The main objective of the integration test is to detect failures resulting from the interaction between software components. In the domain of this dissertation, it will be important to detect failures from the devices' values. At this stage of testing, it is also important to guarantee the flow of information between components.

Within integration testing, there are several levels. Components integration is the first and it will guarantee correct communication between internal components and subsystems. System integration ensures correct interfaces of different systems and between hardware and software.

In integration testing, there are a few strategies that can be followed. The most generic ones are: top-down, bottom-up, ad hoc and backbone integration. In the top-down integration, the test starts in the top-level component. Stubs replace the subordinate components and, iteratively, integration continues throughout the lower-level components. The advantages of this strategy is that test drivers are not needed since the higher level components have already been tested. The disadvantages are the cost of such operation due to the fact that stubs must replace lower-level components. The bottom-up integration starts with elementary system components that only make calls to the operating system. The advantages of this strategy are that there is no need for stubs, but test drivers must simulate higher-level components. In the ad hoc integration, components are tested as they are finished. This method is especially good because every component is tested as early as possible in the development of the final product. Stubs and test drivers are needed. Lastly, in the backbone integration, a "skeleton", or "backbone", is built and the components integrate it as they are finished. With the use of this strategy, components can integrate the backbone at any time, but the process of building such skeleton can be costly.

**System Testing** is the phase of software testing right after integration testing. In this phase, the focus is to ensure that the integrated product meets the specified requirements. System testing is especially important because the system is seen from the perspective of a future user. As such, the solution is validated according to the requirements. Another important factor of system testing is that, sometimes, many functionalities result from the interaction of more than one of the sys-

tem's components. Such functionalities are only seen when the system is assembled. Sometimes, system testing can be misleading mainly due to the fact that the requirements were not gathered meticulously at the start of the development phase, so it is unclear on how the system is supposed to behave.

**Acceptance Testing** is the last stage of software testing. At this stage, which is the only one not under the producer's responsibility, tests are performed before the presentation of the software to the user or customer. It is the only level of testing where the user is involved or they can understand. Acceptance tests are especially recommended if the user and customer are two different entities.

## 2.2 Testing Challenges and Needs for IoT

IoT is a very heterogeneous environment and it brings new challenges, different from the classic software development ones. A great number of developers, who are used to develop more traditional software solutions will most likely underestimate the effort required to test and debug such systems [TM17]. There are also some experiments regarding the test of GUI's in mobile applications with some promising results [CPN14]. According to [Sof], the process of testing IoT systems includes performing end-to-end tests, with the following steps:

- **Functional Testing** - understand the customer's requirements
- **Compatibility Testing** - checking and validation of combination of device's communication protocols and operating systems' versions
- **Usability Testing** - verify the system's ease of use
- **Network Testing** - check the correct functioning with different available networks
- **Security Testing** - verify privacy, reliability and lack of password encryption mechanism within an IoT system. Check the use of security standards
- **Performance Testing** - verify the overall scalability and performance of the IoT system with a focus on power consumption, memory used and capability of switching of network

One of the problems in the IoT domain is the lack of interoperability at the network level. This is by definition the opposite of what such systems try to accomplish. The process of programming such systems is different compared to typical ones, because of the presence of a much smaller computational power and very restricting network related issues.

During the process of developing solutions for IoT, some tests can be performed in device simulating platforms to assess the correct functioning of the device. Although this makes the task of testing a lot easier, it is still important to test the system with real devices due to the combination of both software and hardware [RWBO15].

Present in Fig. 2.2 is the Edge-Fog Cloud architecture. Fog computing paradigm has been proposed, where cloud services are extended to the edge of the network to decrease the latency and network congestion [GVDGB17]. The outermost layer of the Edge-Fog cloud is composed

of a large number of human-operated devices connected via ad-hoc network chains [MK16]. The inner layer is composed of a dense network of Fog devices with high computing power. Since it is a decentralized architecture, the Edge-Fog cloud makes it possible to decouple processing time from network delays by effectively handling processing close to the data generators. Such architecture offers reliable data storage of both raw and computed data at the central data storage located at the core of its architecture.

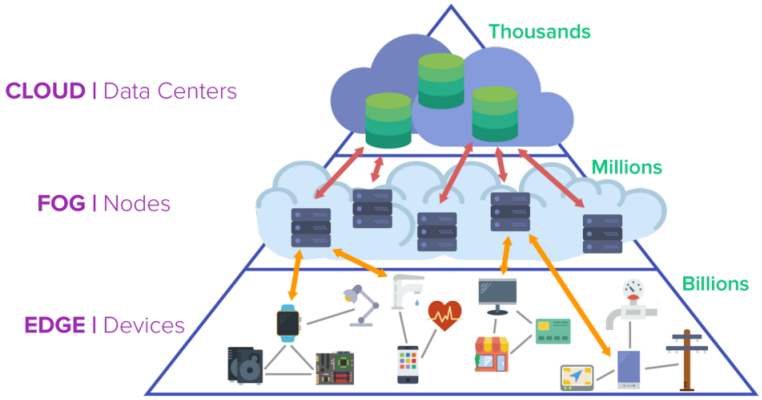



Figure 2.2: IoT architecture of Edge-Fog Cloud
[Spo]

## 2.3 Test Frameworks and Visual Environments for IoT

Nowadays, there is a large number of development and testing tools for IoT. As such, a research was made in order to identify tools to be possible candidates as a basis for this work [DCPF18].

From the analyzed solutions, a few points were taken into consideration. Firstly, and one of the most important topics, is the IoT layer the tool works over. The more layers the tool covers, more complete it is and makes it a better candidate. The second, and no less important than the first is the test levels the tool supports. Then, the programming language. This is an important factor especially in open-source tools due to the fact that it is possible to add extra features. Also, it was imperative to understand the test environment over which the tool would work over. The aim of this dissertation is to work with physical devices and not simulated ones. After, it was important to perceive the platforms that a framework supports. Most of the tools only support a single platform and that is a disadvantage. The idea is to support as many platforms as possible. It was also collected information about whether the tool was developed in an academic environment or a commercial one. At par with the last topic, was also determined whether the license was open or closed. An open tool is more important due to the fact that it is possible to improve it and add new features to it. Lastly, it was analysed whether the tool possessed a visual interface and the domain of such interface.

The tools investigated are presented in this section and a comparative synthesis is presented in the next section.

**PlatformIO [Plu].** It is an open-source Integrated Development Environment (IDE) for IoT solutions developed by PIO Plus. This framework was dedicated to developers wishing to develop simple IoT solutions, with not much detail, but also for big companies who have the intent of deploying large scenarios with a great number of devices. PlatformIO supports multiple platforms and a unit testing system. It works on the edge layer and its tests are run within the physical devices. For an easier start, PlatformIO features a set of already deployed platforms, for distinct IoT devices. The developer can either start with an existing one or set up a new one, with its own devices and operating system.

It is a tool with a lot of focus on development for IoT. It features an IDE that allows the developer to organize the code and libraries. It has a C and C++ programming languages compiler and code linter [1]. It also makes use of a built-in terminal and a serial port monitor. As it is perceptible, this tool is very much focused on the development of solutions for IoT and not testing. In fact, the only testing support is unit testing. It works over a large number of platforms, which makes it very heterogeneous. Also, the way it communicates with devices is by having them physically connected and that's a plus.

Although it is a commercial tool, it is open-source, which means that it is possible to improve its current state. The most important point regarding this tool is the fact that it does not feature a visual interface of any kind. Also, the only available testing level is unit testing. These two topics are of imperative importance for the selection of a tool.

PlatformIO has a large community of developers and moderators that exchange comments and ideas through an official forum and it is available to use in three different licenses - Community, Professional and Enterprise.

**IoTIFY [Gmb].** It is a cloud-based IoT performance testing platform for scenarios with a large number of devices. It was developed by Ternary GmbH. It's a company that focus their development on web applications, cloud platforms, embedded software and network and security. With the development of IoTIFY, their objective was to make the development of IoT solutions faster and easier.

IoTIFY works on the Edge, Fog and Cloud layers of IoT, which is a great advantage. Unlike PlatformIO, this tool allows a much wider range of testing levels. It includes support for unit, integration and system testing. The test environment in IoTIFY is only simulated which is not adequate for this dissertation since one wants to execute tests on physical devices.

Regarding the programming languages, IoTIFY does not mention in which language it is programmed in. Also, it is not perceptible which platforms it supports. IoTIFY is a commercial tool and its license is closed.

[1] Linter: a tool that analyzes source code to flag programming errors and stylistic errors

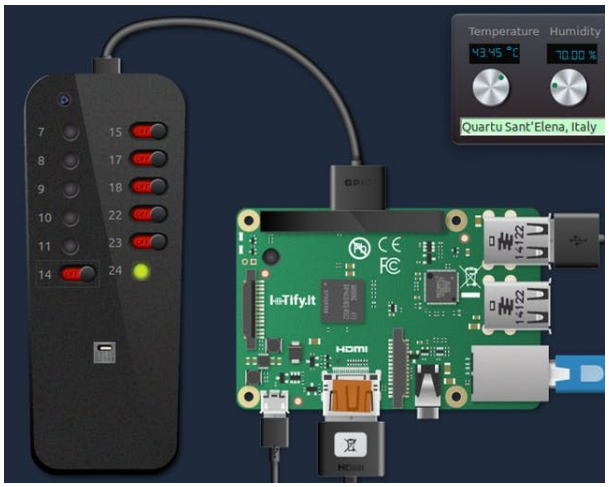



Figure 2.3: Virtual IoT Lab - IoTIFY
[Gmb]

In regard to the visual interface, IoTIFY possesses one with a large number of features. The interface is called "Virtual IoT lab", as represented in Fig. 2.3, and enable the user to simulate a virtual hardware lab. It allows for the customization and demonstration of single components and its interaction with the overall network of IoT nodes or the simulation of numerous IoT devices on large-scale environments, such as cities. It is a very intuitive and complete visual interface since it can simulate single devices or multiple ones both in small as well as very-large-scale environments. Unlike PlatformIO, this framework does not support any online forums so that developers can share their ideas and doubts.

**FIT IoT-LAB [Fac].** It is a very large scale infrastructure with the purpose of testing a large number of small wireless sensors and other heterogeneous communication devices. IoT-LAB was developed by FIT - a Research Infrastructure (IR) by the French Ministry of Higher Education, Research and Innovation. FIT results of a consortium of five institutions of higher education and research that are devoted to making testbeds for network computer communications available to the enterprise, scientific researchers, and educators. All of the software produced in FIT labs is licensed by a CeCILL license.

Like IoTIFY, it supports all IoT layers. Its main purpose is the testing of scenarios as described above and not for the development of IoT solutions. FIT IoT-LAB allows unit, integration and system testing. One important feature of this tool is the use of physical devices, unlike IoTIFY that simulates physical devices in a virtual environment. There is no information regarding the programming languages in which this platform was developed in. Another topic of great importance is the fact that this is an academic as well as a commercial tool with an open license. This means that it is able to reuse, but it is not open-source, so it does not allow for improvement or any addition of features.

Regarding the visual interface, this tool does not possess one, which is a great disadvantage.

This is, in fact, a tool with a lot of focus on testing. They provide a set of testbeds [2], spatially distant, that allow researchers to monitor a large number of important variables, such as energy consumption or various network metrics.

For starters, FIT provides an open GitHub repository where discussions can be made. At the start, they provide a very well structured list of tutorials in which a recent developer may start to learn from.

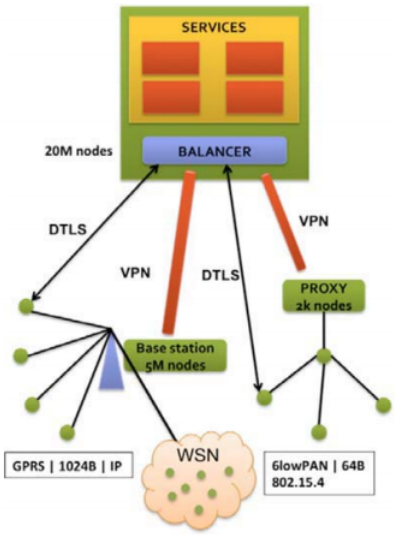



Figure 2.4: MAMMotH architecture
[LODYJ12]

**MAMMotH [LODYJ12].** It is a large-scale IoT emulator for mobile connected devices through General Packet Radio Service (GPRS). The project aims to develop a platform for a large number of devices, whilst emulating a real-world IoT environment. MAMMotH was developed by a team of engineers from the School of Science, Aalto University in Finland. Their main motivation for the development of such tool was due to the fact that they found out that the current solutions are most appropriate for small and medium-scale emulation, however, they are not suitable for large-scale testing that reaches millions of nodes running concurrently.

MAMMotH is a very favourable tool due to, mainly, two aspects: IoT layer and the levels of tests it allows to perform. Regarding the IoT layer, MAMMotH works on all IoT layers. It is not limited by any layer, like FIT IoT-LAB or IoTIFY. Regarding the levels of test, it allows for the testing of integration and system levels, which is only a portion of all levels, but the most important one is present - integration.

On the downside, MAMMotH team did not make public the programming language in which it was written in. Also, the connection to the devices is emulated, not needing to have physical

[2]Testbed: a platform for conducting rigorous, replicable tests of scientific theories and new technologies.

devices to work with. The platforms supported and its license are also two aspects that remain unknown. MAMMotH was developed in an academic environment.

Regarding the visual interface, it is not provided with any information about possessing one. Unfortunately, since this is a tool developed in an academic environment, there are no forums or communities where the developers can have discussions about the tool.

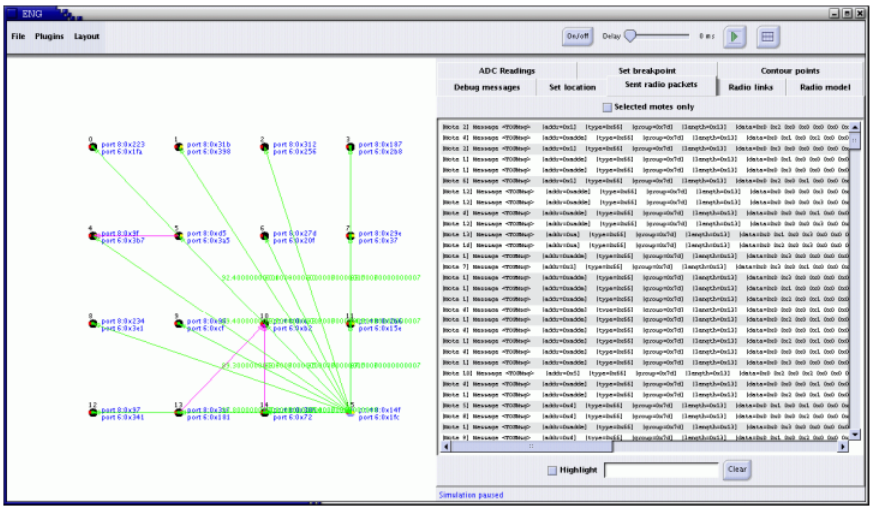



Figure 2.5: TOSSIM visual interface
[LLWC03]

**TOSSIM [LLWC03].** It is a wireless sensor network simulator. It was built with the specific goal to simulate TinyOS [3] devices.

It was developed by three researchers from the University of California, Intel Research, California and Harvard University, Massachusetts. TOSSIM was created with the objective of capturing network behaviour at a high fidelity while scaling to thousands of nodes.

TOSSIM is a tool that is focused on the testing, especially on integration testing which is a plus in the domain of this dissertation, and not in the development of IoT solutions. However, the IoT layers it covers are reduced. In fact, it only supports the Edge one. The framework was developed using Python and C++ programming languages.

It was developed in an academic environment and its license is open to be reused. It also features a visual interface, as shown in Fig. 2.5, for user interaction. Unfortunately, both the GUI and the testing part of the framework are focused on the test of radio connected devices. This means that the domain is distinct from the one which is requested in this dissertation.

Although TOSSIM is a very complete tool, it does not work with physical devices. Instead, it simulates a set of radio connected devices in the interface and further allows to select them for testing. Also, the devices must be running TinyOS operating system in order to be used in this

[3] TinyOS: embedded, component-based operating system and platform for low-power wireless devices

framework. TOSSIM does not feature any support from the developers so that users wanting to use this tool can settle their doubts, unfortunately.

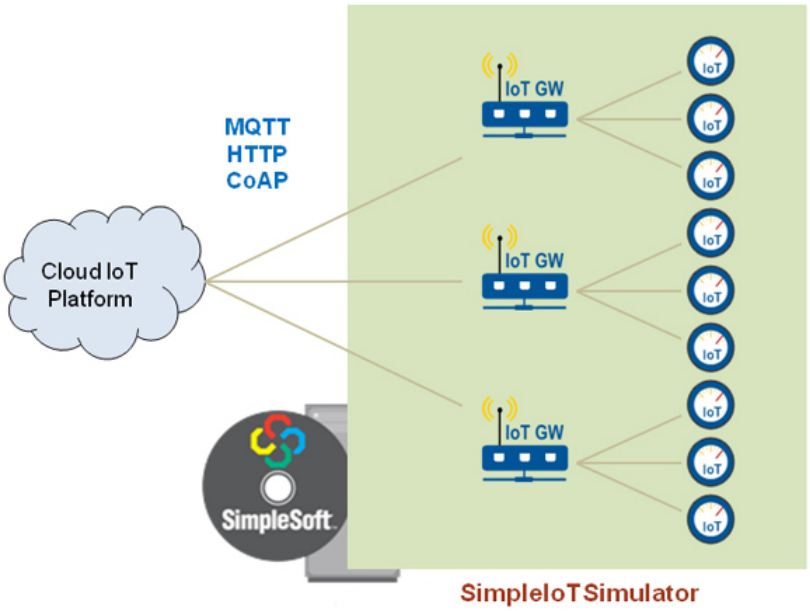



Figure 2.6: General schema of SimpleIoTSimulator
[Sim]

**SimpleIoTSimulator [Sim].** It is an IoT Sensor/device simulator that creates test environments with a large number of sensors and gateways. It was developed by SimpleSoft, a company located in Silicon Valley, Mountain View, California.

In Fig. 2.6 we can observe the high-level architecture of the SimpleIoTSimulator. It makes use of common communication protocols, such as Message Queuing Telemetry Transport (MQTT), Hypertext Transfer Protocol (HTTP) and Constrained Application Protocol (CoAP) to establish communication from the cloud into the framework. The packets arriving from the cloud are targeted to the gateways within the tool. After that, they are distributed to the respective IoT devices.

SimpleIoTSimulator is a framework with a focus in the testing of devices and not in the development of IoT solutions. Within the testing, it is focused mainly on integration testing.

Although it is focused on integration testing, SimpleIoTSimulator also has its drawbacks. It only works in the edge and fog IoT layers. Also, there is no information regarding the existence of a visual interface or its domain. The programming language in which the framework was developed was not provided and the test environment involves using a simulated scenario. That is a drawback since the desired tool would have to support physical devices. Another downside of SimpleIoTSimulator is the fact that there is no information regarding the supported platforms. Also, this is a commercial product and its license is closed. SimpleIoTSimulator does not possess any forum where developers can settle their doubts.

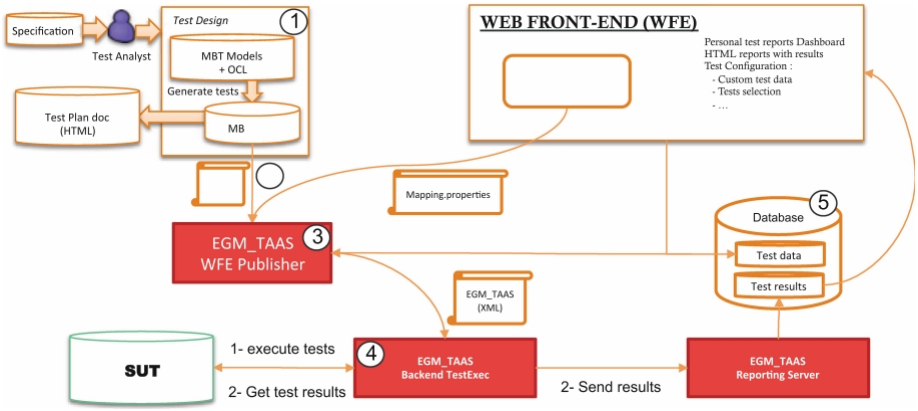



Figure 2.7: General schema of MBTAAS architecture
[ABF+16]

**MBTAAS [ABF+16].** It is an approach that combines Model-Based Testing (MBT) techniques and service-oriented solutions in a platform that allows IoT testing. It was developed using a joined force from Easy Global Market, Université de Franche-Comté and Smartesting Solutions and Services. The tool was originally intended to make the testing an easier task, mainly due to the latest increase in the use of IoT devices and IoT-related scenarios.

MBTAAS is, in fact, a very powerful tool. Its architecture can be visualized in Fig. 2.7. It allows for testing on all levels - Unit, Integration, System and Acceptance. It also features support for all IoT Layers. MBTAAS was developed using OCL (Object Constraint Language), which is not a very popular language. The test environment is on platform and there is no information regarding the number and types of supported platforms for this tool. Although it was implemented by a team of developers from the commercial and the academic scopes, it is considered an academic tool. Unfortunately, there is no information regarding its license.

In terms of visual interface, MBTAAS possesses one so that the user can configure and launch the tests to be executed. The visual interface features a set of predefined tests stored that the user may select from. It is also possible for the user to execute only part of the tests available. The GUI also provides support for test results visualization. Although it is a graphical interface, it still requires some programming and technical knowledge to operate it. The user must input the test configuration. Unfortunately, It does not feature any discussion area for developers to settle their doubts.

**SWE Simulator [GMPE13].** It is a tool developed with the intent of representing multiple types and number of sensors and integrate with a standard sensor database known as Sensor Observation Service. It was developed by a group of four researchers from the Departamento de Comunicaciones of the Universitat Politècnica de Valencia. The main motivation of the team was to develop a tool that would natively interoperate with SWE (Sensor Web Enablement) architecture. Its high-level architecture can be observed in Fig. 2.8.

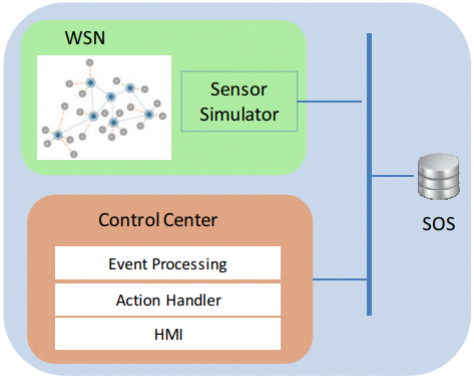



Figure 2.8: General schema of SWE Simulator architecture [GMPE13]

SWE Simulator is a very powerful tool but it is very much focused on testing of wireless sensor networks. As a result, the IoT layer it focuses on is the Edge one. Also, it does not allow for integration testing. Instead, it only supports system testing, which is a downside. Another disadvantage is the fact that it does not work with physical devices. It makes use of simulation in order to operate.

This framework was programmed using XML and Visual Basic programming languages and, as a strong upside, it possesses a visual interface. Although it is a good point, this GUI's only objective is to monitor the small wireless sensors' activity. It does not feature any possibility for configuration. SWE Simulator is a tool developed in an academic environment so it does not feature any forums, or communities, that can help starters to settle their doubts, unfortunately.

**MobIoTSim [PKSL16].** It is a mobile IoT simulator to help developers handle devices without buying real sensors, and demonstrate IoT applications using multiple devices. It was developed by two researchers from the universities of Szeged, Hungary and Antwerp, Belgium. Its main objective is the simulation of devices and the response to critical values read in the sensors.

MobIoTSim is a testing framework that focuses on the Fog and Cloud IoT layers. That is an advantage, although not completely. The ideal tool would support all IoT layers. Also, it supports integration testing, which is the main objective of this dissertation. It was developed in an academic environment, as already stated, and its license is open to reuse. Apart from allowing the testing of multiple devices, also enables the user to customize each individual device, as shown in Fig. 2.9.

Unfortunately, there is no information regarding the programming languages in which it was coded in and, since it only simulates devices, it is a disadvantage for our solution. Also, is not present information regarding the number or which platforms it supports.

In concern to the visual interface, it is present, but as an Android application. MobIoTSim connects the devices to an Android mobile device so that the user can have access to the values being read in their mobile devices. This was a tool developed in an academic environment and


Figure 2.9: MobIoTSim device customization
[PKSL16]

there is no support to any forum or community.

**DPWSim [HLC+14].** It is a framework that helps developers to implement and test IoT applications, making use of Devices Profile for Web Services (DPWS) technology, by simulating physical devices. It was developed by a team of 4 investigators from the Institut Mines-Telecom, France, University of Science and Technology, Korea and Smart Systems Laboratory, France. Although the team involves one investigator from the commercial scope, it is considered an academic developed tool. Its architecture schema is presented in Fig. 2.10.

DPWSim works on the Fog and Cloud IoT layers, but not on the Edge layer. Also, it supports integration tests, which is the objective of this dissertation.

It possesses a few drawbacks. First of all, it is programmed using Web Services Description Language (WSDL) programming standards. Also, its focus on simulating devices, which is not what is desired in this dissertation. Another downside is that it only supports DPWS platforms. One of the key points of this project is that it was required to wire together devices from multiple platforms and programmed in multiple languages.

DPWSim has a visual interface available, although it is out of the domain. This GUI, unfortunately, only provides management and simulation support for DPWS devices. It does not feature any forums or communities for developers to have discussions.

**Atomiton IoT Simulator [Ato].** It is a testing framework that simulates virtual sensors, actuators and devices with unique behaviours, which communicate in unique patterns. It was developed by Atomiton, located in Santa Clara, California, USA. Its main objective is the simulation of IoT

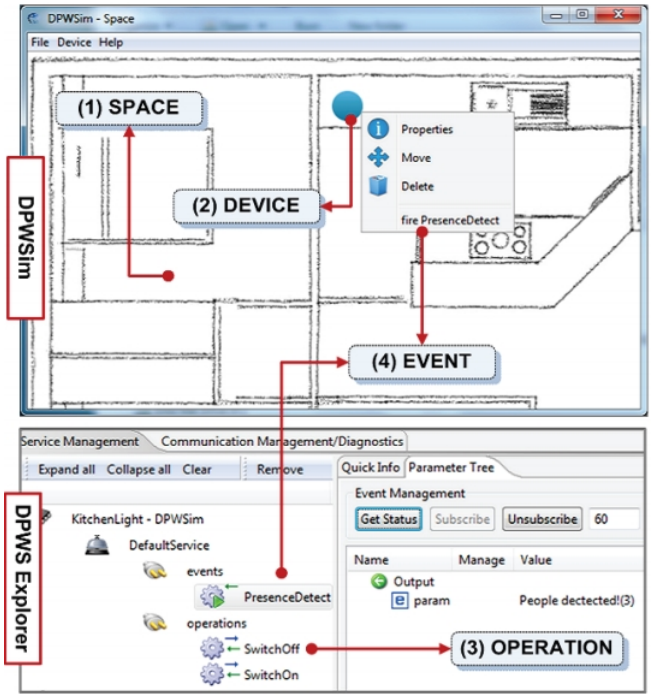



Figure 2.10: DPWSim high-level architecture
[HLC+14]

solutions and interactions within each scenario. It also has the possibility of simulating all kinds of breakdowns, traffic jams or other environmental factors. It is a tool with a lot of focus on the commercial side.

IoT Simulator is a very complete tool. It supports all types of test levels and works on every IoT layer. Unfortunately, there is no information about the programming language it was developed in and its license is closed so without these two crucial factors, it's impossible to add any additional features. The high-level architecture of IoT Simulator can be observed in Fig. 2.11. Also, its focus is on the simulation of devices and, in this dissertation, it is expected to use physical devices. It features a visual interface but only for virtualization of devices. It does not support any forum or community for developers to settle their doubts.

**Izinto** [PLF18], as introduced before, is a pattern-based integration testing framework. Its architecture can be observed in Fig. 2.12. It was developed in a previous dissertation with the intent of solving some of the problems mentioned before. In fact, Izinto abstracts the configuration of a system's components, but it still requires technical knowledge to work with it, since the user must, manually, write the configuration file. In a certain IoT scenario, including a few sensors, with different goals and given a configuration file, Izinto will execute the tests described in the configuration file, with its properties and values. In the end, a report, in JUnit, is presented to the user with the results of the tests that were just executed.

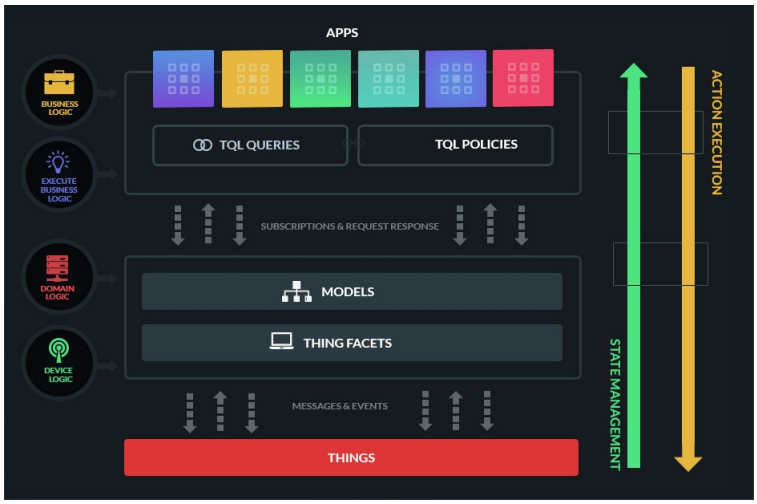



Figure 2.11: Atomiton IoT Simulator high-level architecture [Ato]

We can consider two main modules that we can split Izinto into: test logic module and IoT components module. The test logic module, which was implemented using JUnit, is responsible for the application of the test patterns described in the configuration file. Each pattern is performed in a different method with the JUnit @Test annotation, using JUnit's Assertions. The IoT module is responsible for the necessary communication with the IoT devices in the testing scenario. These allow to configure and control said components, along with a set of classes that represent concepts such as readings and actuator commands, along with alerts and actions. As to input data, Izinto prior to the starting of the integration test interprets a configuration file, which is written in JSON, and describes all the steps and properties of the test to be executed. The JSON file has the structure as shown in the Figure 2.13. The excerpt corresponds to part of the configuration file required for testing the AM-2302 sensor, expected to perform temperature readings every minute and humidity readings every two minutes, with a maximum deviation of 3 seconds and a maximum transmission delay of 2 seconds. Also, the sensor's specification states it is capable of measuring temperatures from -40 $^{o}$C up to 80 $^{o}$C and that it measures the relative humidity from 0% up to 100%.

After this, Izinto will run the tests, as described in the JSON file. In the end, it will output a JUnit report in which the user can visualize the result of the tests, as shown in Fig. 2.14.

As seen, Izinto is an easy-to-use tool, to help a user test its integration scenarios for IoT. As of now, Izinto is a very useful framework, but it lacks some extensibility, adaptability and its architecture is proved not to be the most efficient one.

Izinto is a somewhat static framework when it comes to devices that can be tested. In fact, it possesses the necessary source code to interact with a certain set of devices, but not all. Since devices may come from very different manufacturers, they must be supported by some code in order to make them available for testing. This way, not all devices are ready to be tested, the moment they are integrated into the system under testing.

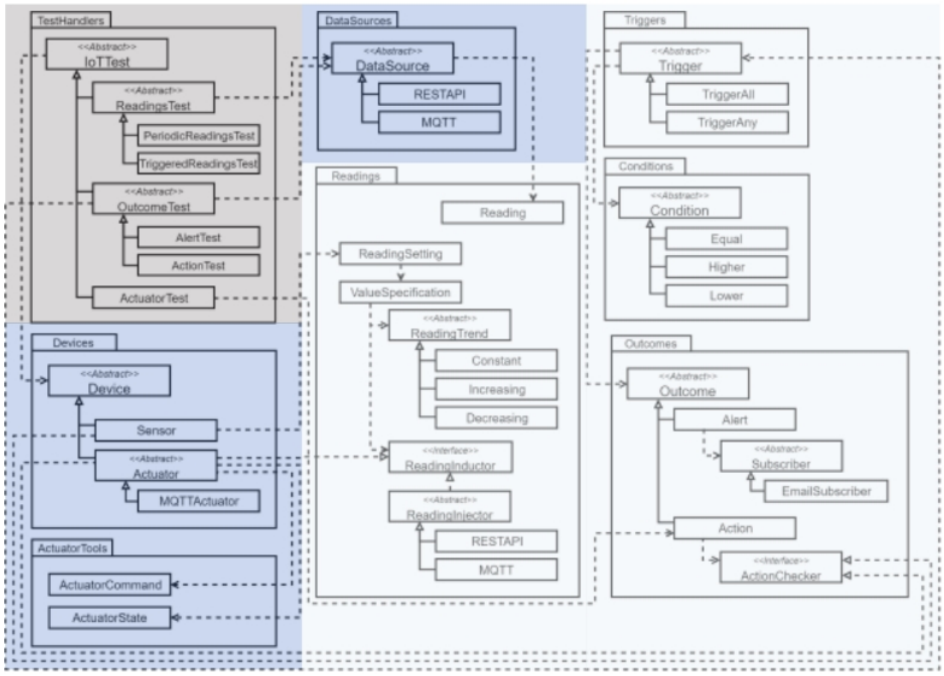



Figure 2.12: Izinto architecture
[PLF18]

The framework makes use of a centralized architecture, although being able to test distributed systems. As already proved, there are already more efficient ways for integration testing, like the hybrid architecture, which combines a central tester with local testers and components under testing [LF17].

Izinto also possesses some strong points, such as its JSON configuration file. In order to start a new integration test, the only file needed is a JSON one, where the tests are described. It abstracts configuration details of the system's various components. This is an advantage since the user only needs to have basic programming knowledge in order to use the framework.

Izinto supports action, alert, periodic readings, user triggered and actuator test patterns. In Fig. 2.14, on par with a result report from JUnit, it is possible to observe the test patterns implemented.

**Node-RED** [Fou] deserves special attention due to the fact that utilizes a flow-based approach. That represents an important factor for this dissertation. Node-Red is a browser-based visual editor that allows a user to connect and wire together online services and Application Programming Interface (API). It makes use of flow-based editing that makes it visually easy for an average user to create simple, or more advanced, connections between the referred entities as shown in 2.15.

Node-Red was developed on Node.js. This is an advantage due to the fact that Node.js is an open-source, JavaScript interpreter and makes use of its event-driven and non-blocking model. Also, Node.js possesses an open repository for its packages with more than 225,000 different modules. This makes it easier to integrate with a great number of technologies and to solve simple

```
{"type": "Sensor",
 "specs": {"id": "AM-2302", "readingSettings":
          [{"type": "Temperature",
            "expectedInterval": 60000,
            "acceptableDeviation": 3000,
            "acceptableDelay": 2000,
            "valueSpecification": {"minimumValue": -40,
                                   "maximumValue": 80}},
           {"type": "Humidity",
            "expectedInterval": 120000,
            "acceptableDeviation": 3000,
            "acceptableDelay": 2000,
            "valueSpecification": {"minimumValue": 0,
                                   "maximumValue": 100}}]
}}
```

Figure 2.13: Input data for Izinto
[PLF18]

problems in a very simple and efficient way.

Node-Red, in fact, is a very complete technology but in its own domain, which is a visual interface. In fact, Node-Red does not provide any support for testing, especially in IoT. It was not developed with the intent to support the testing of any level. Anyhow, we can retrieve a few good practices from this tool. Its visual interface is well aligned with what we aim to develop - a flow-based visual interface with the possibility of customization of each node.

At par with Node-RED, **easyedge** [Dom] deserves special attention due to the fact that it utilizes a low-code solution approach. Users with no technical knowledge are able to configure the testing scenarios and the framework handles to rest of the process. easyedge is an IoT platform that allows the connection of multiple devices. It allows developers to rapidly deploy their IoT scenario without the need for programming. Like Node-Red, it also possesses a visual interface that enables the user to visually design their scenario the application scenario (Figure 2.16).

One of the most important points regarding easyedge is the fact that it allows for the communication of several types of devices through a unique platform the team has developed. This platform even allows for devices to communicate through the most popular cloud services.

easyedge is, in fact, a very good starting point for this project. It uses flow-based programming, same as the Node-Red, in a very user-friendly way, in order to connect multiple devices. It lacks, however, a few important topics. easyedge does not support testing, which means that we would have to use another tool for the testing part. Another downside is that it provides support for the Edge IoT layer and requires a license in order to work with it.

## 2.4 Synthesis

From the tool analysis made, we can conclude that there are a few tools that can potentially be used for IoT development and testing. Izinto, Node-Red and easyedge are not present due to being either focused on testing only, in case of Izinto or being focused only on a visual tool. In

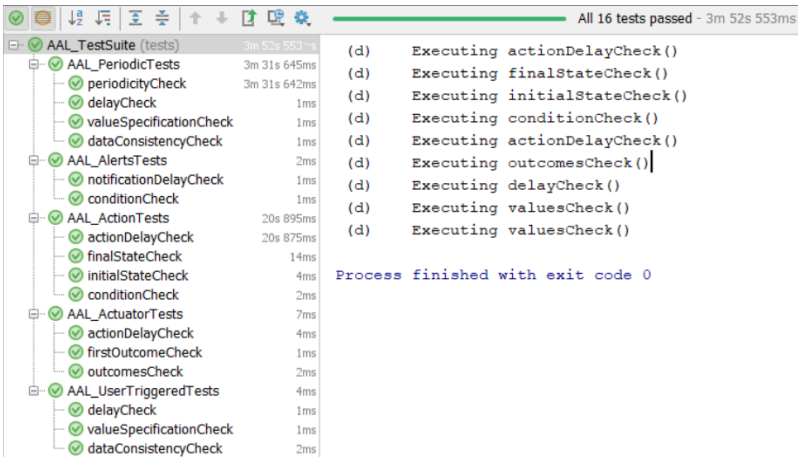



Figure 2.14: Results of an Izinto test run
[PLF18]

table 2.1 is presented a condensed analysis of some tools. There are, in fact, a large number of tools that provide support for all IoT layers, which is an advantage and that is aligned with the objectives of the dissertation. There are also a few tools that don't support integration testing, which is a downside. One of the most important factors we can point out is a large number of closed license tools. This is a very negative factor, mainly due to the fact to not be able to improve the current state of the tool and the addition of new functionalities. Another important factor is the test environment. In this dissertation, it is expected to be working with physical devices and not simulated ones. Actually, only one tool provided support for physical devices. Most tools also lack the extensibility needed to work over multiple platforms, being, most of them, platform-centred. The objective of this dissertation is the possibility of aggregating numerous platforms, from different devices. The most crucial point to be evaluated was the existence, or not, of a visual interface for easier interaction. Being this factor the most important, we could conclude that most tools did not provide the necessary UI or it was not the most suitable one for the domain required. Some focused on very large-scale environments, with devices of a single platform.

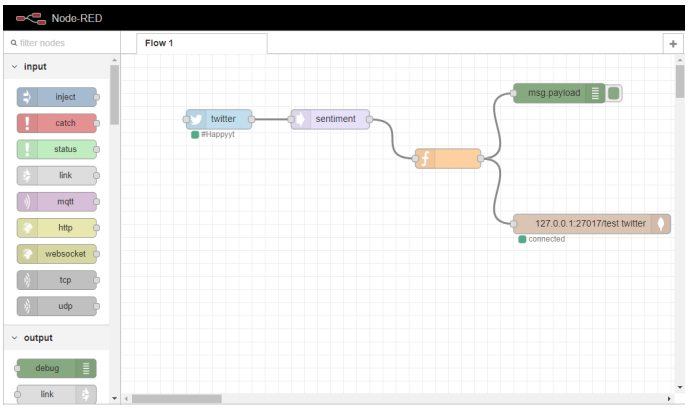



Figure 2.15: Node-Red workplace example
[Fou]

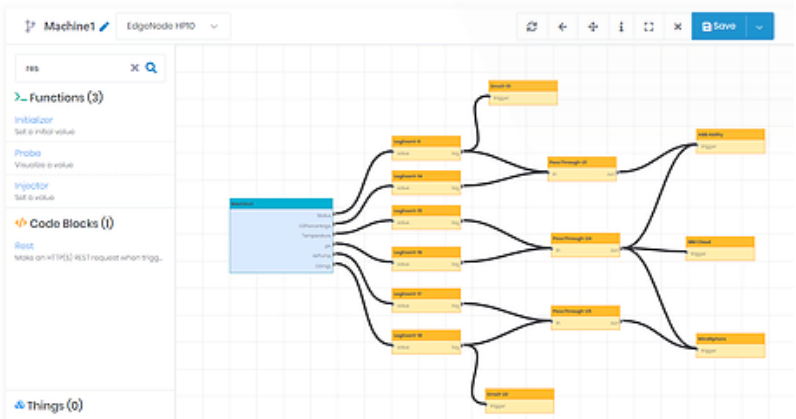



Figure 2.16: easyedge workplace example
[Dom]

| Tool | IoT Layer | Test Level | Prog. Lang. | Test Envir. | Supp. Platforms | Scope | License | UI | UI Domain |
|---|---|---|---|---|---|---|---|---|---|
| PlatformIO | Edge | UT | C/C++ | Device | Multiple | C | Closed | N | - |
| IoTIFY | All | Any | N/A | Simulator | N/A | C | Closed | Y | Only for virtualization of devices |
| FIT IoT-LAB | All | Any | N/A | Physi. Testbed | Multiple | A / C | Open | N | - |
| MAMMotH | All | IT, ST | N/A | Emulator | N/A | A | N/A | - | - |
| TOSSIM | Edge | IT | Python/C++ | Simulator | TinyOS | A | Open | Y | Only for radio connected devices |
| SWE Simulator | Edge | ST | XML, Visual | Simulator | SWE Standard | A | N/A | Y | Monitoring of wireless sensors' activity |
| MobIoTSim | Fog, Cloud | IT | N/A | Simulator | N/A | A | Open | Y | Android application |
| DPWSim | Fog, Cloud | IT | WSDL | Simulator | DPWS | A | N/A | Y | Simulation of DPWS devices |
| SimpleIoTSimulator | Edge, Fog | IT | N/A | Simulator | N/A | C | Closed | N | - |
| Atomiton IoT Simulator | All | Any | N/A | Simulator | N/A | C | Closed | Y | Only for virtualization of devices |
| MBTAAS | All | Any | OCL | Platform | N/A | A | N/A | Y | Requires technical knowledge |

Table 2.1: Synthesis of the analysed IoT tools for both testing and visual environment.

# Chapter 3

# Solution Design and Implementation

This chapter describes the design and implementation of the solution. Furthermore, it will be made reference to the way it interacts with the tool for integration testing for IoT - Izinto [PLF18], described in chapter 2 2.

The objective of this dissertation is the development of a low-code visual interface that allows a user to design its system under tests, composed of several sensors and actuators, and to test its integration. The tool developed will attempt to diminish both the time needed to configure a test as well as allow a user with no programming knowledge, to test its IoT scenario.

## 3.1 Architecture and Technologies

For the development of the solution, it was decided to use a JavaScript framework - Node.JS [Noda] with the intent of integrating both the visual interface and Izinto [PLF18]. Apart from Node.JS, it was used a flowchart design framework, also written in JavaScript - JointJS [Joi] for easier and faster development of the solution. In the following chapters, it will be detailed all the implementation architecture, with a greater focus on the architecture of the visual interface.

From the analysed tools in chapter 2, JointJS was chosen mainly due to the fact of being a very extensible framework, with a great number of features and functionalities, but also being very well documented and popular made it an obvious choice. It is important to notice that this tool is not present in chapter 2, since it is a purely visual and diagram design tool and it is not focused on the development, or testing, for IoT. It possesses a large number of online communities where developers can settle their doubts.

The High-level Component View of the test system is shown in Fig. 3.1. The area displayed with the yellow background in Fig. 3.1 was the one introduced to the already existing tool for testing, allowing the visual configuration of the test scenario.

In fact, there is a certain dependency of the implemented solution on the existing tool in order to run the test configured in a graphical way. The developed solution allows for the visual definition of a scenario with a set of devices (sensors, actuators, notifications, etc.), their interaction and the refinement of test parameters. After, the configuration file is delivered to the Izinto testing

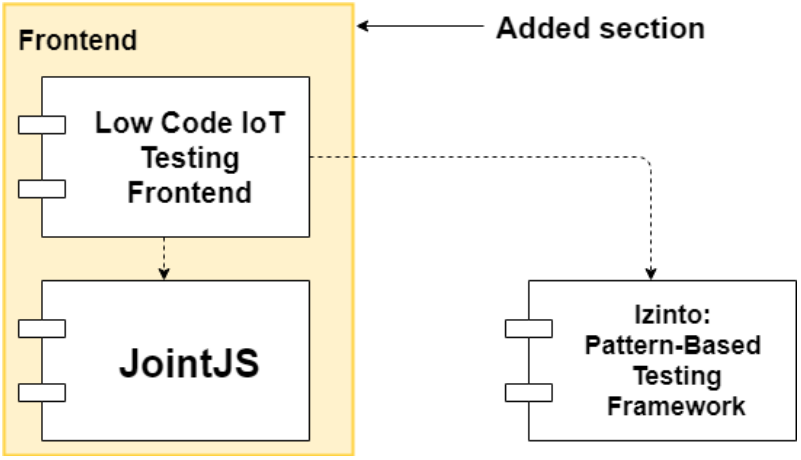



Figure 3.1: High-Level Component Diagram of the Frontend.

framework. Upon finishing the test execution, Izinto will return, on par with a JUnit test report, a set of more specific messages, so that it becomes clear where errors might have occurred and be able to display such problems to the user, in the visual interface.

In Fig.3.2 is possible to observe the dataflow view of the solution implemented. The diagram is accompanied by a sequence of steps, represented by numbers along the scheme, that shows the path that the data follows across the whole architecture.

Following the sequence of actions for test execution and further test results observation, let us start from number 1 - Visual test configuration. At this stage, the user will set up its test scenario with a set of devices and their parameters. As devices are being generated and linked in the workspace, a set of test patterns, which are available out-of-the-box, are being suggested to the user. Internally, the application is interpreting the current workspace and informs the user about patterns possible to be executed. Once the user has finished its scenario definition, it must input one last parameter, which is the test time. After that, simply executes the tests.

The next step (2) is the generation of a textual test configuration. At this stage, the application will check which tests the user is able to run, through an algorithm which was developed to suggest tests, in regard to the designed scenario. The algorithm will associate the respective devices with each test and generate a JSON file, according to Izinto standards. For each pattern test, it is generated a different configuration file, although they are all ran at the same time. Izinto will execute the tests by communicating with the devices within the system under testing (sensors, actuators, notifications, etc).

Upon test completion (3), Izinto, more specifically JUnit, will return a report in the format of text. Such report was altered from the original Izinto files for both easier and more comprehensive understanding of the test failures, and successes, in the web application. Such report (4) will be interpreted by the logic module of the web application and will demonstrate to the user the errors that may have occurred. There are three main ways of showing the results to the user. Firstly, the user will have "drawn" beside each pattern a red cross (in case of failure) or a green checkmark (in case of success). Second, these same figures will be displayed inside each sub-test (each test is

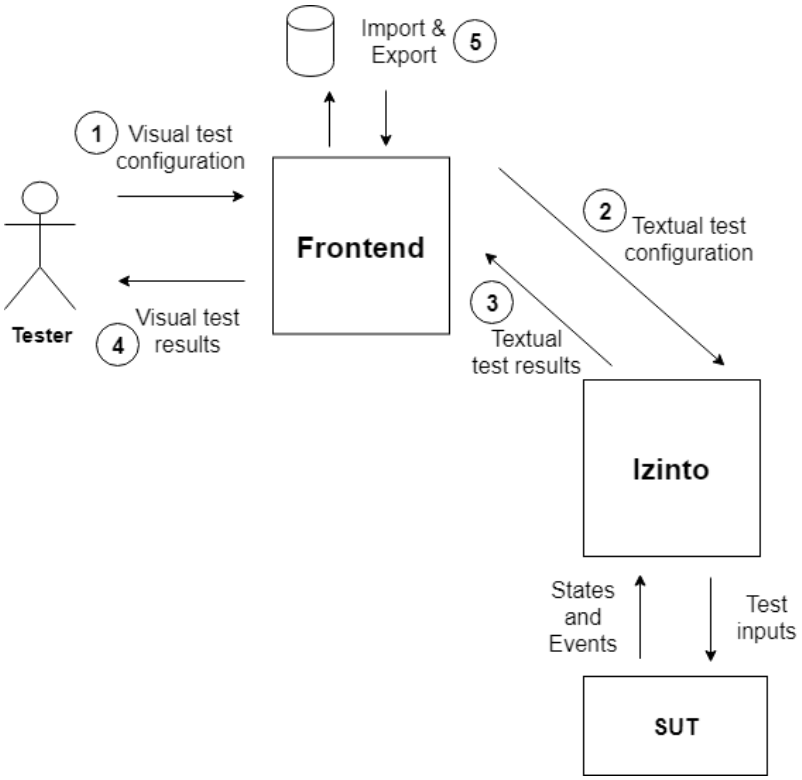



Figure 3.2: Dataflow View of the Solution

divided into smaller and more precise tests). Lastly, the elements of the workspace will be painted green, red or grey in case of success, failure or not tested, accordingly.

For a better user experience (5), the application allows for exporting and importing of scenarios. After a user fills its workspace with its testing scenario with a set of sensors and actuators, it may wish to save that scenario. There is a feature that allows a user to download a JSON file that represents the current scenario in the workspace and all devices' parameters. Later, when the user wishes to resume the test scenario design, or simply run the tests again, it can import the file into the web application. An interpreter will read the file and automatically generate the scenario as it is described in the JSON file.

## 3.2 Main Functionalities and UI Design

In Fig. 3.3 we can observe the entry point for the solution developed. In this picture, it is possible to distinguish five main parts, or areas, each regarding different functionalities or objectives.

On the left, there is the presence of a toolbox in which the user can create blocks. Blocks, which will be later explained with more detail, are abstractions of physical sensors, actuators, applications and e-mail notifications. Each block has a small and simplistic form for the user to fill in and specify the parameters of the abstraction it represents and also for the test to be executed.

On the right (2), there is a place where it is possible to find the test patterns the tool supports and allows testing for. There are five patterns available - Action, Alert, Periodic Readings, User

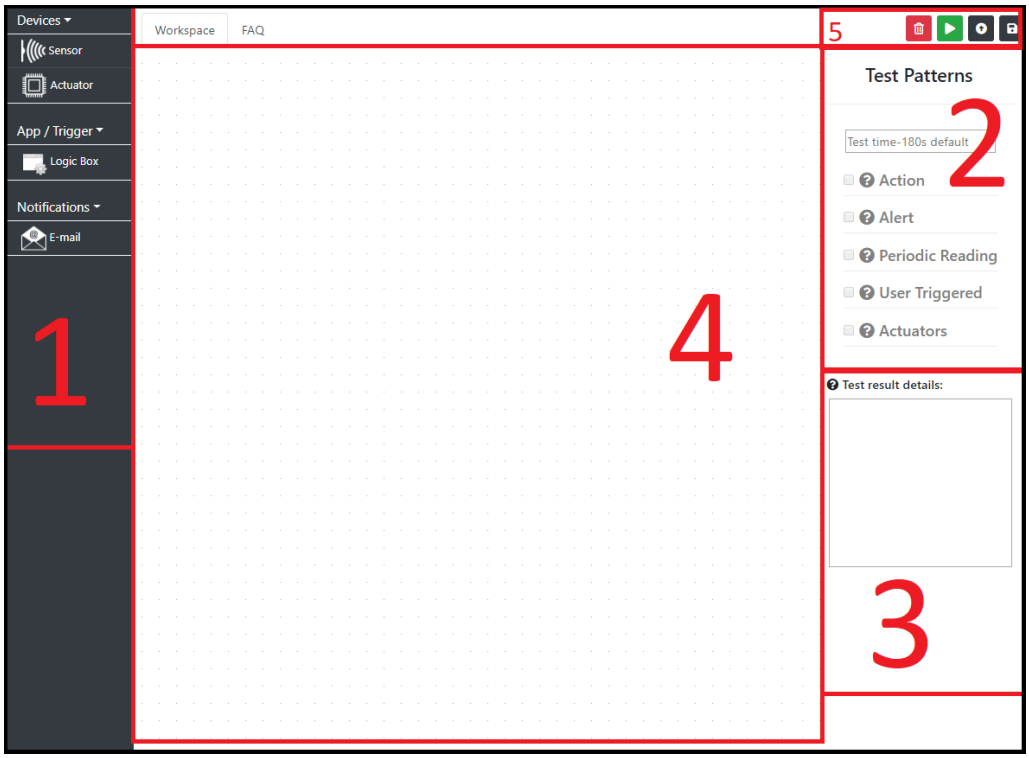



Figure 3.3: Workspace for scenario definition and pattern test suggestion to be applied to each scenario. The image is already split into the most important parts.

Triggered Readings and Actuators. In the Action pattern, it is tested if the readings of a sensor trigger the execution of an action. In the Alert one, it is tested if the readings of a sensor trigger the execution of an alert for the e-mail specified. In the Periodic Readings pattern, it is simply tested if a sensor, or multiple ones, are capable of producing readings with a certain time interval. Regarding the User Triggered pattern, it has the purpose of testing whether the sensor is capable of triggering a reading with the interaction, or the manipulation, of a user. Lastly, in the Actuator pattern it is tested if an actuator is capable of changing its internal state (from ON to OFF and vice-versa). In this area is also possible to choose which test pattern the user wishes to execute, and later observe the results obtained.

Bellow the Test Pattern area (3), there is a text box where a user can observe, with greater detail, the results of an executed test. The text box displays text as the user clicks in the sub-tests, present below each "major" pattern.

There are four buttons present, on top of the Test Pattern area (5), each with a different objective. The "Garbage Can" button has the purpose of deleting all blocks, and links, that may be present in the workspace, clearing the workspace. The "Play" button, in green, has the purpose of executing the tests selected. The "Floppy Disk", as the icon suggests, allows the user to store, in its computer, the current scenario, by downloading a JSON file representing the actual configuration. Lastly, the "Upper Arrow", enables the exact opposite of the save button. It allows the user to upload an already saved scenario into the workspace.

In the middle, there is the workspace in which the user is able to move, connect and edit the blocks' properties and test parameters.

## 3.3 Visual Definition of Test Configurations and Scenarios

One of the main features of the tool developed is the flow-based scenario design. The user can describe its testing scenario, using blocks and connections between them in order to replicate a physical connection of sensors. The sections necessary for the design of the scenarios are the toolbox which includes a set of blocks the user can create. In the following sections they will be detailed and explained.

The **Tool Box** is a spot located on the left of the workspace where it is possible to create blocks for the workspace. Inside the toolbox area, there are three different sections: Devices (abstraction for sensors and actuators), App / Trigger (abstraction for applications that may trigger certain actions in regard to the values read in the sensors) and Notifications ( a way to alert the user of the reaching of certain values).

In the **Sensor**, the user is asked to fill in a small form, represented in Fig. 3.4, with parameters for both testing and own details of the sensor. Among a large number of readings most sensors can perform these days, it has been chosen to implement only a few: Temperature, Humidity, Air Quality (CO2), Diastolic Pressure, Systolic Pressure and Weight. The process of choosing the reading types for the sensors allows the user to choose multiple readings for the same sensor, for example, temperature and humidity. After, the user is requested to fill in a few parameters regarding the test: Acceptable Deviation, Expected Interval of Time ( interval of time to perform readings of the sensor) and Acceptable Delay (time took to get the readings from the sensor). As the reading types checkboxes are being filled, a new section of the form is displayed to the user in the format "Reading Type" Values Specification. This section has the purpose of guaranteeing that the sensor's readings stay within certain specified limits. Also, the user may choose a reading trend: Increasing, Decreasing and Random, according to a certain value trend and the user may also choose if this reading is triggered by human interaction with the sensor (User Triggered Tests). This section of the form is "repeated" for all sensor's readings chosen. Finally, the user clicks in the button "Add Block" and it will show up in the workspace. In Fig. 3.5 it is represented the format and design of the sensor when the block is generated and put into the workspace.

On the top left corner of the block the user will find a red cross which will eliminate the block from the workspace. When the block is generated, the user may wish to edit the initial configurations. It is possible to edit them by double-clicking on top of the element, while it is on the workspace. A similar form to the one used to create the element in the first place is displayed to the user making it possible to alter the original parameters. This form is displayed in the middle of the screen, as a Modal dialog.

In the **Actuator**, the user must fill in a different form from the Sensor. Actuators have two purposes in the scope of the application - they are abstractions for actions that are triggered when

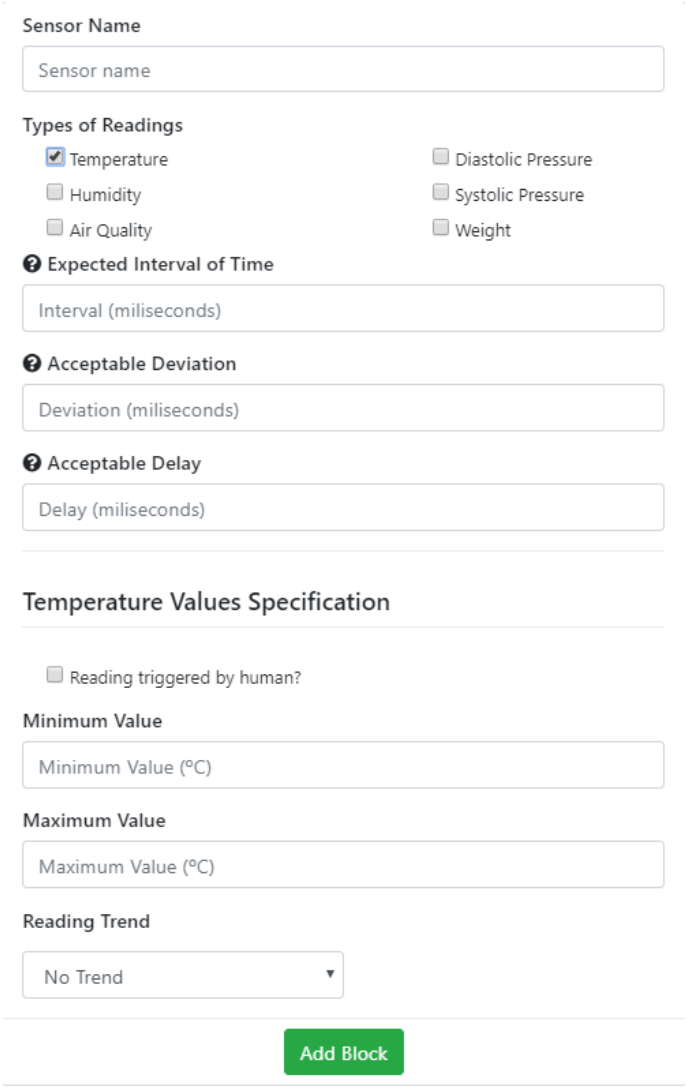



Figure 3.4: Form displayed when the user selects the Sensor from the Devices sub-menu.

certain values are met by applications and sensors, but also can be tested in regard to their internal state (ON/OFF). Starting with testing the actuator's behaviour, the user is presented with the form represented in Fig. 3.6. Firstly, and with less importance, the user chooses the name to designate the actuator and its purpose, as already stated. After that, a new section is added to the form - Actuator Test - where the user will indicate the parameters for the test. The number of state changes represents the number of times the user wishes the actuator to change from ON to OFF and vice-versa. Next, the user must input de IP address of the MQTT (Message Queuing Telemetry Transport) [MQT] the actuator will be listening from. Also, indicate the topic from which it will be subscribed to. The next step is to input the acceptable delay for the actuator to respond to commands, indicate the command which will make it change its state, from ON to OFF. The last two inputs regard the argument to join the state changing command which will turn ON and OFF the actuator. Finally, the user clicks in the button "Add Block" and it will show up in the workspace.

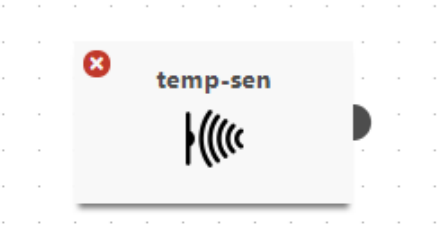



Figure 3.5: Block generated after the user click in the button "Add block" in the sensor creation form.

In fact, actuators can have a different purpose, other than testing their behaviour. They can serve as an abstraction for an Action, for example, turn on an air conditioning, or a lamp. When the "Purpose of Actuator option is selected, a slightly different form is displayed. For the purpose of executing an action, different parameters are required for the user to fill in. Firstly, the user is required to indicate the MQTT broker address and state topic, like in the previous section, in order to establish communication with the actuator, in this case via a message queue. Then, the user is required to describe the time which the actuator will take to respond to an event, the message it will trigger upon the action execution, and, like in the test of the actuator's behaviour, the command to toggle ON/OFF and the argument to execute such action.

As well as the sensor, the actuator also possesses a red cross on the top left corner of the block, when it is generated with the intent of eliminating it from the workspace. Also, if the user double clicks on it, an edit form will be shown and the user may change its original properties.

The next tool in the Tool Box is the Logic Box and it works as an abstraction for a real application, with logic, for the purpose of executing an action, or notifying the user when certain conditions are met. Upon clicking the Application, from the sub-menu App / Trigger the form represented in Fig. 3.7 is displayed. As usual, the user is required to set a name for that box/block. Afterwards, the user must choose whether the Logic Box will be an outcome for an Alert or an Action. It is possible to do that by setting the next item of the form - Type of Outcome - as an Action or an Alert, accordingly. Next, the user should introduce the maximum amount of time (delay) the application can take to check whether the condition was met. A description is the next requirement of the form. The description is important to describe, briefly, the intent of this application, for example - "Temperature reached the value of 30 $^{o}$C". The next step is the initial and final state of the actuator, in case of an Action. If the intent of the test is to test if the air conditioning was turned on after the temperature reached a certain value, then the initial state could be OFF, but the final state, if, in fact, the temperature reached such values, should be ON. The last item in the form is the specification of the triggering action which may have two choices - TriggerAny - which means that the action/alert will be triggered when any of the conditions are met or - TriggerAll - which means that the action/alert will be triggered only when all the conditions are met. After specifying all the parameters, the user simply clicks in the "Add Block" button and the block is generated into the workspace.

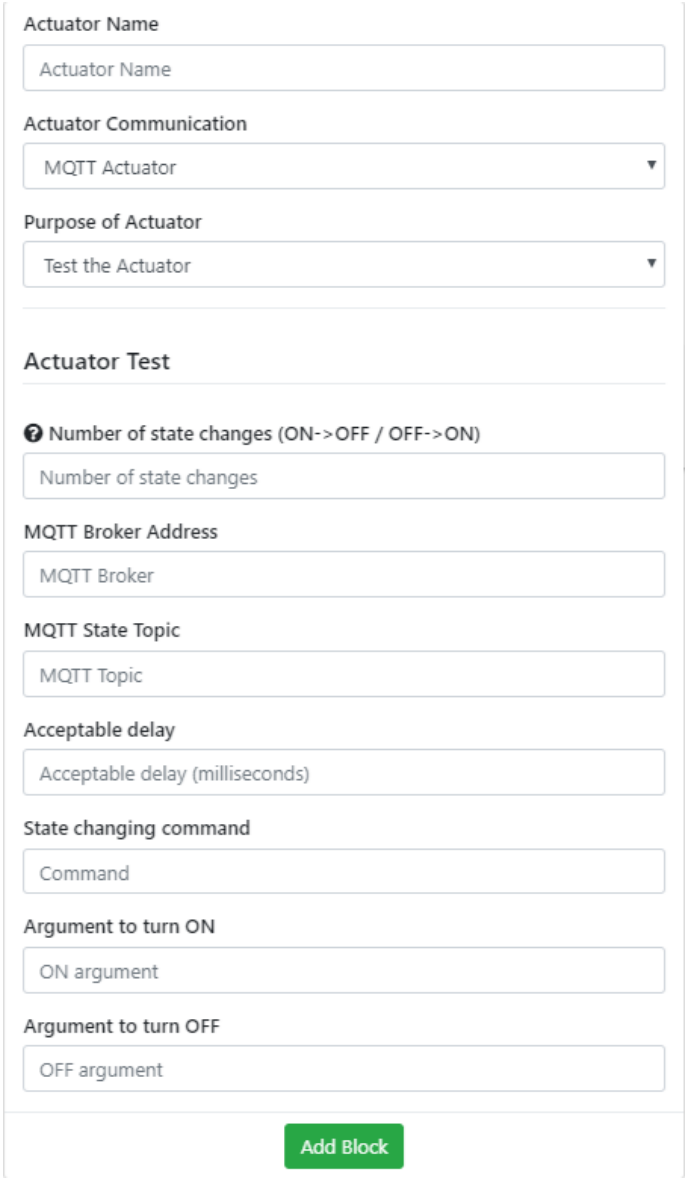



Figure 3.6: Form displayed when the user selects the Actuator from the Devices sub-menu and chooses "Test the Actuator" in the purpose of actuator item.

In order for the user to specify the conditions for the triggering of action or alerts, the logic box must be linked with a sensor. In the following example let us assume that the sensor performs readings of temperature and humidity. In Fig. 3.8 it is possible to observe the form shown to the user as the new link is created. As observed in the image, the sensor and the logic box are connected and the form now has the purpose of letting the user choose the intervals of values of the readings performed by the sensor that will trigger an action or an alert. The user may input minimum, equal or maximum values. Each is represented as a condition. The conditions may be read as follows (and is also present as a placeholder for each input): "Trigger outcome when reading is lower/equal to/higher than a certain value".

The last tool present in the Tool Box, and in the sub-menu "Notifications" is the E-mail. The

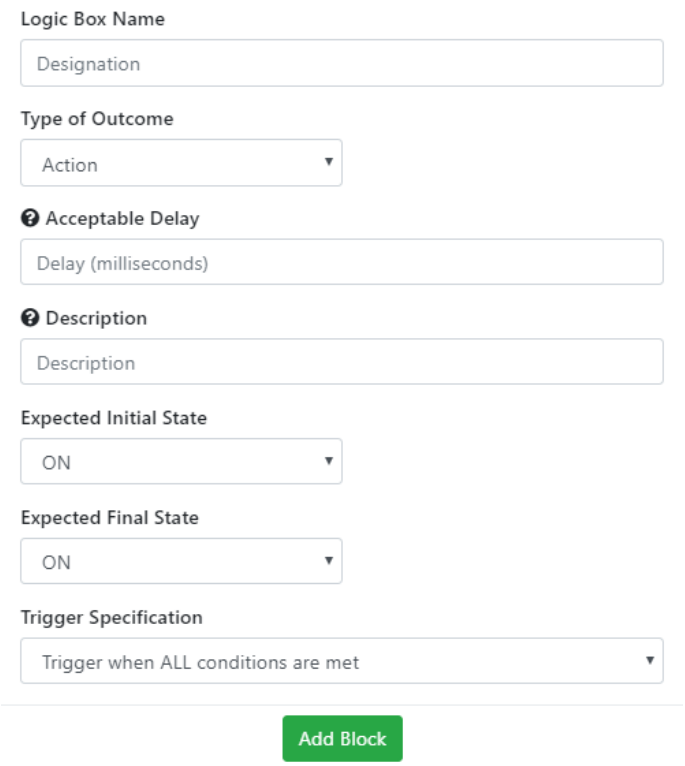



Figure 3.7: Form displayed when the user selects the Logic Box from the App / Trigger sub-menu.

form that is displayed to the user when it clicks in the block is represented in Fig. 3.9. This block will be responsible for the execution of an alert, in this case, the sending of an e-mail. In the form, the user is presented with three simple fields to fill. Firstly, the block name, as usual. Next, the user should introduce the e-mail address where the notification will be received in and the password to this e-mail. The password is required only to guarantee that the e-mail was, in fact, sent to the e-mail address just indicated. After specifying all the parameters, the user simply clicks in the "Add Block" button and the block is generated into the workspace.

The connections, or links, between block are very relevant, especially if referring to the action or alert test patterns. In these two cases, the way blocks are connected are of imperative importance to the test suggestions algorithm. The algorithm will evaluate the designed scenario by looking at the way blocks are connected and from those connections evaluate if these patterns are possible to be executed. For example, in the action pattern, the algorithm will check if in the scenario is present a sensor, which is connected with a logic box and the logic box connected to an actuator. Only in this case, it will make the pattern available to be executed. In the case of the other three patterns, the connection is not relevant, since only one block is required to use them.

## 3.4 Test Selection and Execution

The test execution phase includes the suggestion and selection of test to run and the passing of test parameters to Izinto.

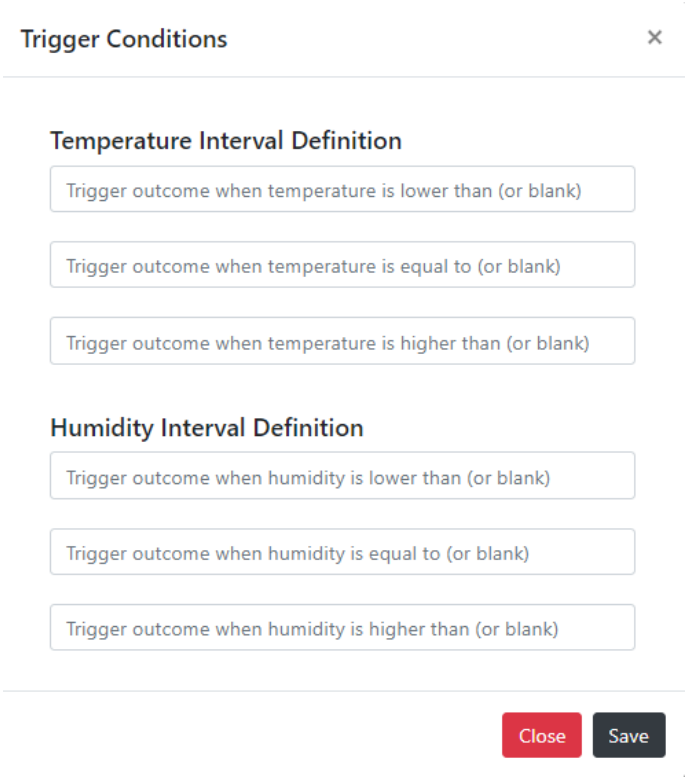



Figure 3.8: Form displayed when the user connects a Sensor to a Logic Box. The program detects which readings the sensor performs and asks the user to set them. These parameters can be changed by entering in the edit form of the Logic Box, by double-clicking it.

In the **Test Pattern Area** the user is presented with the test patterns the tool supports - Action, Alert, Periodic Readings, User Triggered Readings and Actuators test. In Fig. 3.10 is displayed the section of the interface that is referred. Here is the place, within the interface, where the user can select which test patterns to run, according to the design currently in the workspace and their connections. As it is possible to observe in Fig. 3.10, there are patterns that are somewhat faded in comparison to the Periodic Readings one. This is due to the fact that the interface implements a logic module that, at each workspace modification, interprets the current scenario and suggests the user tests that can be run. The current state of the workspace is the temperature and humidity sensor linked to the Logic Box. At this stage, only periodic readings can be performed. If the user would like to perform an action or an alert, would have to link an actuator or an e-mail block to the Logic Box. This way, the action or alert patterns would become possible to "manipulate".

Again, as represented in the image, each pattern is associated with a '?' icon. When the user hovers its mouse over this icon, it is presented with some guidelines and purposes regarding the test. Just after the name of the pattern, there is present, if not faded, an arrow pointing down, which allows the user to perceive which sub-test will be run on each pattern. In the case of periodic readings, in 'periodicityCheck' it will be verified if the readings are performed within a certain periodicity. The 'delayCheck' verifies if the readings are performed within a certain delay. The 'valueSpecificationCheck' verifies whether the values read by the sensor are within the limits

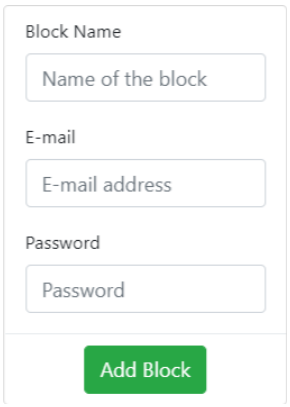



Figure 3.9: Form displayed when the user selects the E-mail from the Notifications sub-menu.

established by the user upon creation of the sensor. Most of the remaining test patterns have either the same sub-test or a set of the ones just described.

Lastly, every test, apart from choosing which one to run, needs one last parameter - test time. This parameter will determine for how long the testing tool will run the test. By default, if not set, the test time is set to 180 seconds, as perceived in the image.

One of the most import aspects of the implemented solution is the capability of suggesting test patterns that can be executed, regarding the scenario designed in the workspace. In fact, the framework that was used to develop the solution, JointJS, was very useful in the way that it allows for consulting the current scenario state, storage of data inside the blocks and their connections.

This way, it was developed an algorithm that is executed when there is a change in the workspace, being the deletion, or creation, of an element, the deletion, or creation, of a link or upon the editing of block properties. This algorithm will attempt to find, in the scenario, flows that represent the five available patterns. In Fig. 3.1 it is present the required blocks, and connections ("->"), to enable each available test pattern.

In Fig. 3.11 it is possible to observe the test pattern suggestion in action. In the scenario, it is present a flow for an alert, which involves a sensor, a logic box and an email block. The sensor is making readings that will trigger the sending of an email when the values read reach a certain point. As described in the image, if the user wished, it could run the "Alert" pattern test. In the event of deletion of one of the integrating part of the alert flow, the test suggestion would also change. In the same image, down, we can see that it was deleted the logic box, connecting both the sensor and the email block. In this case, the test suggestion algorithm will disallow the user to

| **Test Patterns** | **Activation Conditions** |
|---|---|
| Action | Sensor ->Logic Box (Action) ->Actuator |
| Alert | Sensor ->Logic Box (Alert) ->Actuator |
| Perio. Readings | Sensor (1+) |
| User Triggered | Sensor (1+) (Human Readings) |
| Actuator | Actuator (1+)(Test) |

Table 3.1: Conditions to activate each test pattern

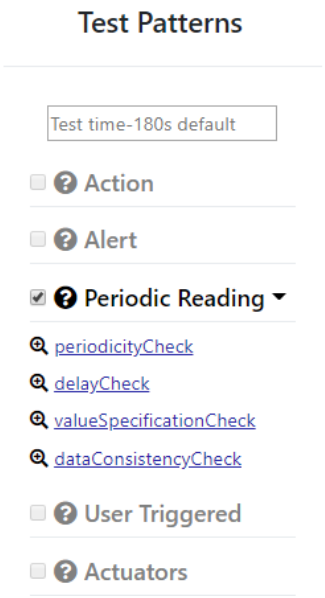



Figure 3.10: Test pattern area. In this section the user may select tests to run and observe the results.

execute an alert test, but allow a periodic reading one due to the presence of the sensor.

So, in order to run the tests, the tool must, firstly, gather which tests the user wishes to run. The user can enable and disable tests in the test pattern area. After, and as already described in the previous section, the test configuration is sent to the server, as a JSON object in an HTTP packet. In that HTTP packet, it is also present, in the headers, five variables, each corresponding to each test pattern and may take the value of "True" or "False". Finally, and since the testing tool has a separate JUnit test for each pattern it is executed a system call to run the tests selected. Izinto will execute the tests as described in the JSON configuration file and interact with the physical devices, present in the SUT, and run the test.

As observable in Fig. 3.12, during the time indicated by the user in the Test Time input, there will be both a spinning wheel and a progress bar so that the user can, firstly, perceive that the test is running and perceive the state it is at any time. Also, the button for test execution is disabled, not allowing for the execution of more than 1 test per user at a time.

The **warning box**, as it is possible to observe in Fig. 3.13 is an indication for the user when some action the user performed could lead to an error and alerts it In Fig. 3.13 the user tried to run the test, but none is selected. There are, across the entire solution, a series of warning boxes that alert the user for either mistakes or guidance to execute some tasks.

## 3.5 Interpretation and Visualization of Test Results

The visualization of results can be split into two different sections of the interface. Firstly, it is displayed as green "Checks" or red "Crosses" on the right side of each test pattern. As illustrated in Fig. 3.10, where the available test is the periodic readings one, there is present a small arrow

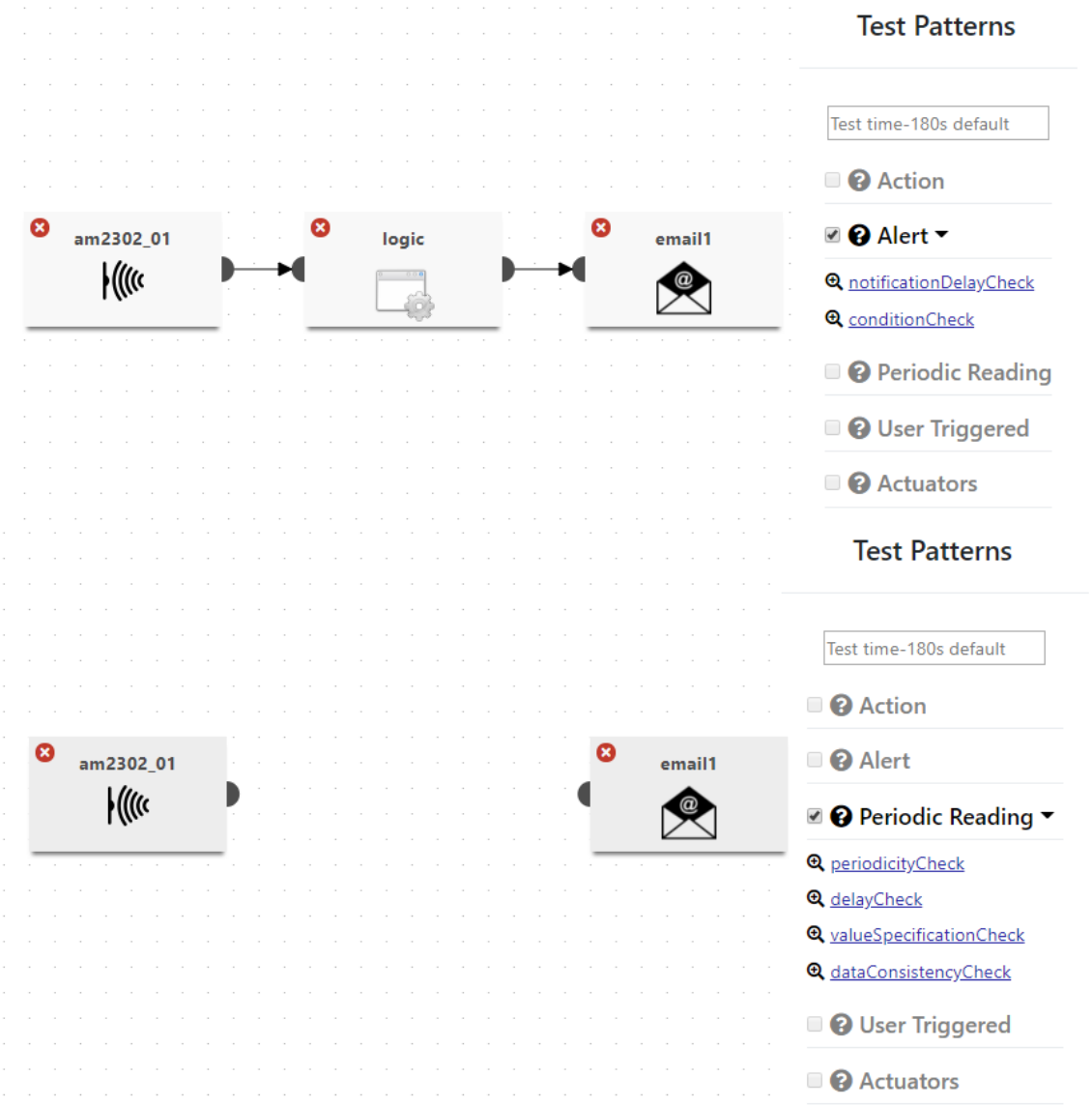



Figure 3.11: Example of a scenario with an alert pattern flow and the alert test pattern being suggested to the user, on the right.

pointing down. Upon clicking in this arrow, it will be open a set of sub-tests regarding the main one. For each sub-test, and after test execution, there will also be displayed the check, or cross, if either the test succeeds or fails. In Fig. 3.14 we can observe the results of a test which had some failures, but also some successes. In this case, it was executed an action test, which involves a sensor, a logic box and an actuator. As observable, the sensor took the green colour, the logic box stayed grey and the actuator got the red colour. This was a design decision to represent the success, in case of the sensor that performed readings correctly and within delay and deviation set by the user and failures, in case of the actuator, that did not change its internal state upon having received an order to do so by the application.

The **Test Detail Area** is especially important so that the user can understand the reasons that

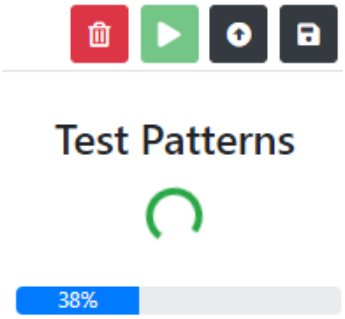



Figure 3.12: During test execution, underneath the title of the section "Test Pattern" it is displayed both a spinning wheel and a progress bar so the user can observer the test progress.

triggered the failure of a test. In Fig. 3.15 it is possible to observe an example of a test failure in which the user forgot to connect the sensor to the gateway. This led to an error triggered by JUnit and interpreted in the interface displaying the message "Expected to find readings from sensor | <name of the sensor>".

## 3.6 Extension Modules for Izinto

### 3.6.1 E-mail Notification System

For the purpose of validating the solution developed, in the Alert pattern it was expected to send an email when certain conditions were met.

As such, and with the objective of enriching the solution, it was developed an e-mail notification system, using NPM (Node Package Manager) [NPM] package, called Node-mailer [Nodb].

The way this notification system works is by listening to the MQTT for temperature and humidity values read by the sensor. Once the test starts, it is notified by the front-end all the necessary

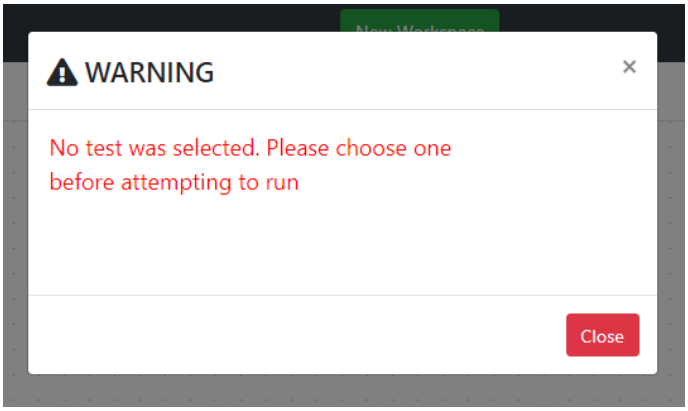



Figure 3.13: Warning box implemented to help guiding the user towards the desired actions. This is one example out of a significant number of different warnings implemented.

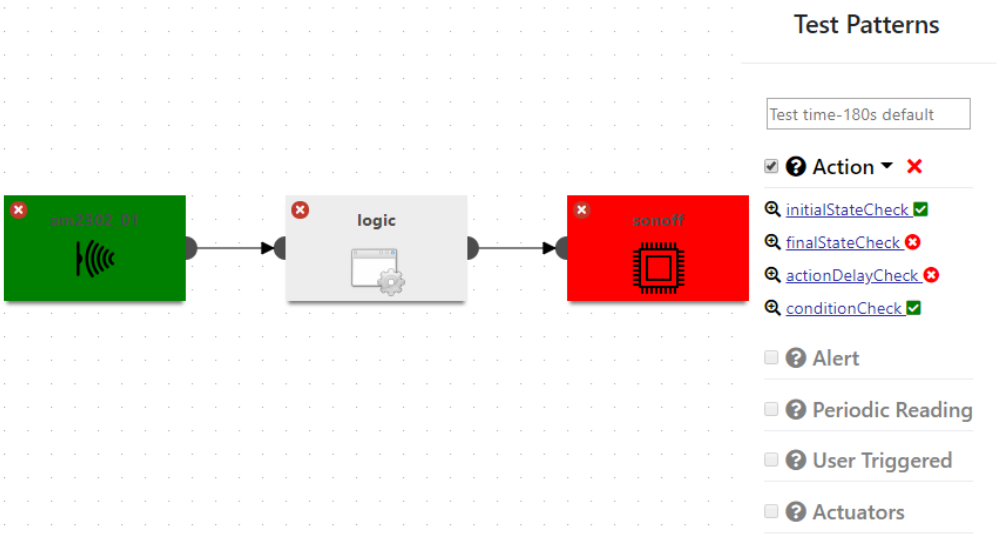



Figure 3.14: Example of a test result with some successes, but also some failures.

information for sending the email (conditions, subject of the email which is the description the user input, etc). When, and if, the conditions are met, it sends an email.

In order for the Alert pattern to pass the tests, it must be used an email with certain permissions. In the case of Gmail, it is necessary to have allowed access for "less secure apps" to enter the email account. Then, Izinto, will check in the inbox for an email, within the timestamps of the start and finish of the tests, with the subject being equal to the description, present in the Logic Box.

### 3.6.2 Connection to Withings API

With the attempt to get readings from a Nokia weight scale, through API call, which has now a new platform called Withings, it was developed a module that allows obtaining the readings. Although a certain effort was put into this, in the end, it was unsuccessful due to the format Izinto receives the readings, among a few other details.

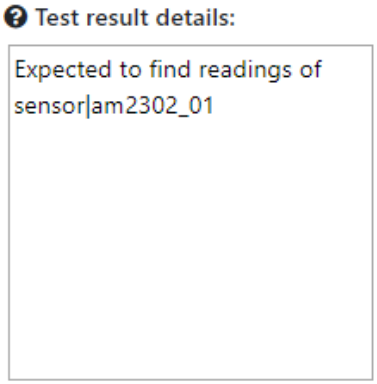



Figure 3.15: Text box that will display details associated with a sub-test failure. When the user clicks on a sub-test, it will display its error.

Although it has no impact in practice and for the validation of the solution, it was decided to include this part since it was spent some time to achieve, at least, the get of readings from the Nokia weight scale.

# Chapter 4

# Validation

In order to validate the solution it was done an usability test. The method chosen was the DECIDE framework [PRS02] which has a certain sequence of steps, observable in Fig. 4.1, that will be detailed next.

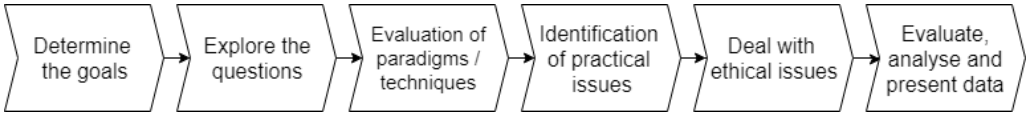



Figure 4.1: The steps of the DECIDE framework for usability test.

The first step of the framework is to determine the goals. In this phase, it is questioned what are the high-level goals of the evaluation and the target audience. It has also the purpose of fine-tuning an interface or even to inform how to following versions of a product should be changed. Goals should "drive" an evaluation.

The next step is to explore questions. In order to make goals an operational force, there are a set of questions that satisfy them must be identified. Such questions can be broken down into more specific sub-questions to make the evaluation more specific and with the objective of assessing concrete problems. The process of creating sub-questions from general, major questions can carry on until they satisfy the goals.

Following the exploration of questions, there is a step in which it is dealt with the evaluation of paradigms and techniques. During this stage, it is determined the kind of techniques that will be used. Also during this stage, it is considered the practical and ethical issues and trade-offs to be made. For example, if the chosen equipment or the appropriate technique is too expensive, so compromises are required.

The fourth step in the DECIDE framework is the identification of practical issues that may occur during the execution of the usability test. The issues that may be identified regarding the users, facilities and equipment, schedules and budgets and evaluator's expertise of the application under test. These issues may involve the adaptation or substitution of the current technique. In the case of the users, they represent the most important aspect of the whole evaluation test and must be chosen with certain care. For example, there may be the necessity of choosing users with different

level of knowledge on a certain area of expertise, different number of men and women, different age range, cultural diversity, etc. Another issue regarding the users is how they will be involved in the tasks. The choosing of the time of each task is important to determine time limits. If a user is having difficulty finishing a task, he/she can get anxious and influence the rest of the test. Regarding the facilities and equipment, in case the experience is being filmed it is important to think about the number of cameras to be available, for example. Also, some users might feel a bit uncomfortable being filmed whilst performing the test, which may influence the results. In terms of schedule and budget constraints, it is important to keep it real and within budget. For example, for a usability test, it may seem good to have 20 users available. However, if it is required to pay them, that can become a costly experience. Also, in terms of time, it is important to keep things within the schedule. In terms of the evaluator's expertise, it is necessary to find a person with enough knowledge of the interface so that he/she can, in fact, help the user through the test. Also, if it is supposed to use statistics, a statistician should be involved before and after the test, if appropriate.

The next step is the decision on how to deal with ethical issues. The data collected during the usability test must not be associated with the users unless given permission to do so. They have the right to privacy. Aspects such as health, employment, education, financial status and address should be confidential.

Following the last step is the evaluation, interpretation and presentation of the data. During this stage is it made a choice of the evaluation paradigm and techniques to satisfy the goals and the identification of the practical issues detected so they can be resolved. Also, decisions are made regarding how the data will be handled. For example, how to analyse it and present it to the development team or how the data will be treated statistically. A usability test should also have a certain reliability. This means that if a different evaluator, or researcher, performing the same test obtains the same results. We are in the presence of a reliable test. A good test should also be valid. Validity is concerned with whether the evaluation technique measures what it is supposed to, regarding both the technique itself and the way it is performed. The bias is the measure of distorted values. Without noticing, the evaluator may be failing to take into consideration certain types of behaviour, thinking they are not important. On the other hand, interviewers may also be influencing the responses from interviewees by the tone of voice or facial expressions. The scope of a usability test refers to how much the findings of a study may be generalized. Finally, the ecological validity on how the environment in which an evaluation is conducted influences or distorts the results obtained.

## 4.1 Planning and Execution

The main objectives of the realisation of the usability test were to assess to quality of the solution developed and to understand if it is able to respond to the problems stated. Synthesizing, the research questions are as follows:

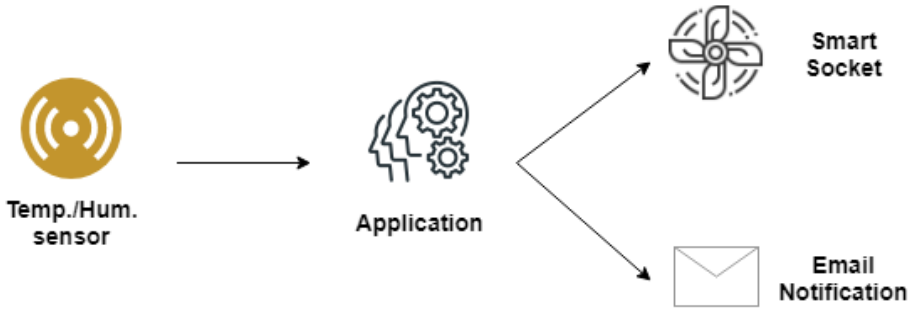



Figure 4.2: Schematic vision of the system under testing.

- RQ1: Do users find it easy and pleasant to create and execute automated tests for IoT systems using the developed solution?
- RQ2: Regarding RQ1, are there differences between users with a low and high technical background?
- RQ3: Are users able to quickly create and execute automated tests for IoT systems using the developed solution?
- RQ4: Regarding RQ3, are there differences between users with a low and high technical background?

The most important aspects described in the research challenges was the possibility of a user with low to no programming knowledge be able to use this tool in order to test a certain system under test with a collection of sensors, applications and actuators. Secondly, it was also necessary to facilitate the process of test definition, execution and interpretation of results.

The test took place during the first week of June.

In order to help identify the quality level of the solution and the utility, some metrics were chosen. Firstly, it was decided to collect the timestamps for tasks. Secondly, it would be gathered the task completion rate and finally, data regarding user satisfaction, made at the end of the usability test.

The choosing of participants was split into two parts. In the first, there was an attempt to get participants with lower technical programming and testing knowledge. By doing so, it would be possible to assess the tool's capability of being used by users with low skill in the area. Secondly, there was an attempt to gather users with higher technical skill. This way, it would be possible to understand if the tool was useful, even for this type of users.

The execution of the test was composed of the developed solution and the system under test. The solution is a web application and the user would access it via a computer with an internet connection. The test was executed in a lab which simulates a smart house. The system under test, possible to be observed in Fig. 4.2, is composed of two parts. Firstly, the temperature and humidity sensor, which was connected to the Raspberry Pi. Secondly, a smart socket, an actuator connected to the lab's WiFi and to the MQTT broker, also set up in the lab. The sensor performs temperature and humidity readings, which are evaluated by an application. This application can

trigger the sending of an email or toggling of an air conditioning when the values reach a certain point, set by the user.

The usability test was composed of five tasks. The complete guide is present in the annex. The tasks were chosen with the purpose of covering most of, or all, the use cases of the developed solution, which are the test patterns. The tasks also had a set of steps for the users to follow. Such steps will diminish in size and detail, as the user completes them in order to understand the intuitiveness of the project.

The first task has the objective of understanding the import functionality, providing the user with the first contact with the selection of test patterns and how to run the tests. Lastly, after the test is executed, the user can observe the test results, useful for the rest of the test experiment. This first task tests the periodic readings of the temperature and humidity sensor.

The second task had the objective of introducing the user to the configuration of an element, or block, namely the actuator, and to learn how to parameterize both the test and the elements. This test has the objective of testing the actuator successful change of internal state, the switching from ON to OFF, a certain number of times.

The third task is similar to the first one, the testing of the sensor's capability of generating periodic readings, only this time the user has to parameterize the test and sensor's inputs instead of uploading an already designed solution. With the execution of this task, it is intended for the user to have his/her first contact with the parameterization of a sensor. This task is similar to the first, but instead of importing an already designed scenario, the user sets up his own.

In the fourth task, it is first introduced the concept of linking of elements. In this task, the user is asked to create a scenario to allow the test of the toggling on of an air conditioning in the event of high temperature. The user is asked to create a sensor, capable of reading temperature, a logic box, which is responsible for setting up the conditions for the action to occur. Lastly, the user creates an actuator and sets up its parameters. In this task, it is intended for the user to have his/her first contact with the effect of connections between blocks and perceiving the data flow between all blocks involved. In Fig. 4.3 there is present the three forms that are required to be filled in in order to create each block for this task.

The fifth task has the objective of testing the correct sending of an email when the temperature read by the sensor reaches a certain value. The main idea of this test was that users could reuse the scenario from the previous task, making very small changes to it and perform this last task. Also, it would be possible to present the users with the warning messages, developed to help them have a better user experience, for example when trying to connect the logic box from task 4, which was set to "Action", instead of "Alert".

## 4.2 Results and Discussion

The users who participated in the usability test are mostly finalists from the informatics and engineering course from the Faculty of Engineering of the University of Porto. This implies, at least, five years of contact with a technical environment, comprising familiarity with both programming

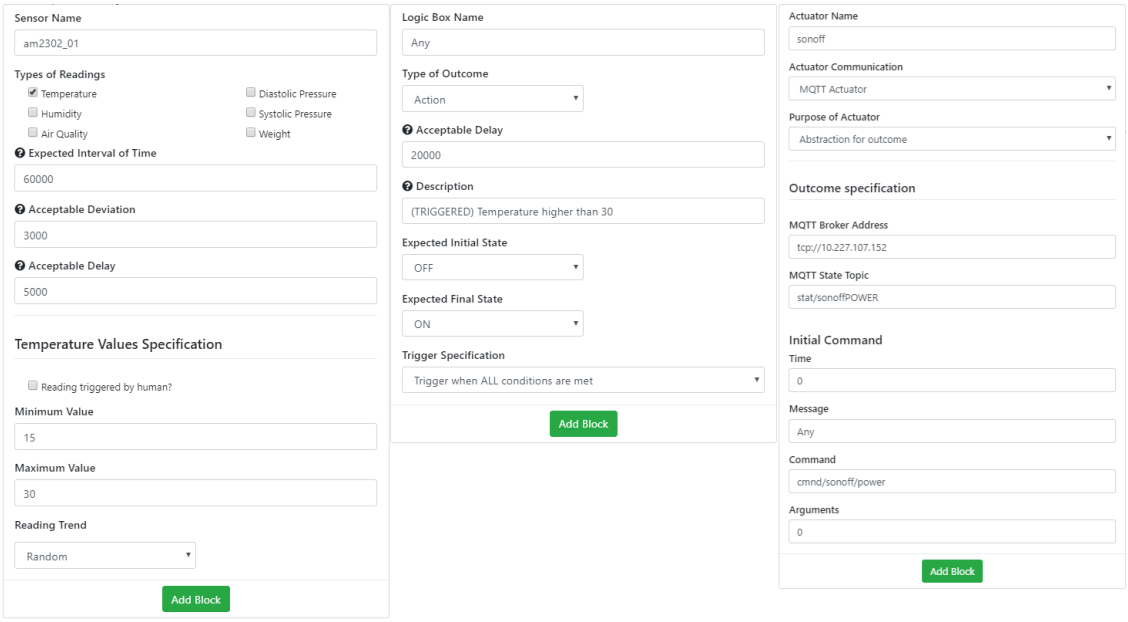



Figure 4.3: Example of the filling in of the forms required for all blocks for the task 4

and testing of software. Regarding the users with a lower technical background, they come from a professional school, enrolled in a practical course on informatics.

The characteristics of the users involved in the usability test are present in Fig. 4.4. From left to right, and top to bottom, in the first graphic is possible to distinguish users in years of contact with programming. In the second graphic, it is perceptible to understand that the users are quite dispersed in terms of IoT knowledge. In the third graphic, the answers get more clear and it is possible to distinguish two groups of users - one with very low technical knowledge and a second one with higher expertise. Also, in the fourth graphic the same idea is present but in terms of testing software-based systems.

| | Question | Global Average | Higher Skill AVG | Lower Skill AVG | T-Test |
|---|---|---|---|---|---|
| **Task 1** | Easiness | 4,91 | 4,88 | 5,00 | 0,351 |
| | Pleasantness | 4,73 | 4,63 | 5,00 | 0,197 |
| **Task 2** | Easiness | 5,00 | 5,00 | 5,00 | - |
| | Pleasantness | 4,82 | 4,75 | 5,00 | 0,17 |
| **Task 3** | Easiness | 5,00 | 5,00 | 5,00 | - |
| | Pleasantness | 4,82 | 4,88 | 4,67 | 0,605 |
| **Task 4** | Easiness | 4,91 | 5,00 | 4,67 | 0,423 |
| | Pleasantness | 4,73 | 4,75 | 4,67 | 0,837 |
| **Task 5** | Easiness | 5,00 | 5,00 | 5,00 | - |
| | Pleasantness | 4,91 | 4,88 | 5,00 | 0,351 |

Table 4.1: Grouped tasks' questionnaires data (Scores range from 1 to 5)

As previously stated, the users were divided into two groups, one having lower skills (participants 1 to 3) and the other having higher skills in programming (participants 4 to 11). In order

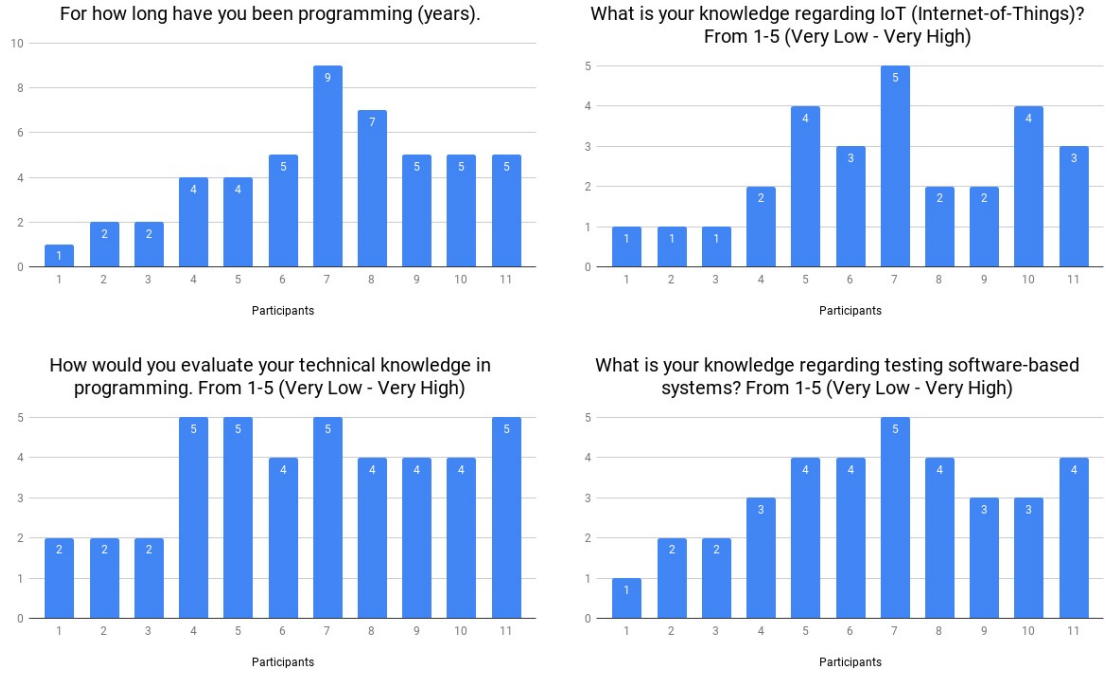



Figure 4.4: Answers obtained in the initial questionnaire with the objective of classification of users in regard to their technical knowledge

to try to answer the research questions to check if there are statistically different results from the two groups, it was made an analysis. Presented in Table 4.1, there is the data collected from the average of scores the users classified each task. Inside each task, it is divided into easiness and pleasantness of execution of that task. The global average, as the name suggests, to the average of all users, without splitting them into the low skill and high skill groups. The "Higher Skill AVG" and "Lower Skill AVG" refers to the average scores of the users with higher technical knowledge and lower skill knowledge, accordingly. The column "T-Test" is the result of computing the t-test for the difference between the two means [Wel38]. By observation of the results obtained, it is possible to perceive they were, in fact, very good. In the global average, all the values are very close to the maximum, in both easiness and pleasantness to perform the tasks and to test an IoT scenario. The T-Test values are all above 0.05 (for a confidence level of 5%), which means that the differences of means are not statistically significant.

In Table 4.2 is presented the times per task by the users. The table aggregates the times from

| | **Average Time** | **High Skill AVG** | **Low Skill AVG** | **T-Test** |
|---|---|---|---|---|
| **Task 1** | 0:33 | 0:29 | 0:46 | 0,063 |
| **Task 2** | 1:46 | 1:43 | 1:53 | 0,587 |
| **Task 3** | 1:43 | 1:38 | 1:55 | 0,091 |
| **Task 4** | 3:58 | 3:51 | 4:18 | 0,540 |
| **Task 5** | 1:38 | 1:10 | 2:52 | 0 |

Table 4.2: Grouped tasks' times (minutes:seconds)

the global average time of all users, the average time per task by the users with higher skill and the average time per task by the users with a lower skill in programming. As expected, the users with a higher skill were, in general, faster than the others. Also, it is important to point out that, although there is a time difference between the two, that gap is not that big. Actually, considering the lack of skill from the group, the times are quite similar. The T-Test regarding the times registered, refer, as well as in the previous analysis, to the difference between the two means of the group with a higher and lower technical background. The T-Test values are most of them above the value of 0.05 (for a confidence level of 5%), which means that the differences of means are not statistically significant. In the case of task 5, the difference is statistically different. The huge time difference between the two groups in task 5 is due to the fact that some users realized the scenario from the previous task could be adapted to the new task with only small changes. Although being statistically significant, it can be perceived that the time required to execute it is somewhat small for the task of testing an IoT scenario.

### 4.2.1 Users Suggestions and Feedback

During the usability test, and also in the end, the users were asked to provide feedback regarding both the tasks and the general aspect of the developed solution. This is a very important aspect since it allowed to get some insight regarding the user experience and general intuitiveness of the application. In the following list are present the most criticised aspects of the application:

1. **Non-user-friendly error messages** - upon test failing the messages present in the Test Detail Box were difficult to understand since they had a very technical aspect
2. **Small test detail area** - the text box, which was unresizable, was very small for the size of the error message
3. **Information of unit on form fields** - some form input fields would not indicate which measure they would. If it was perceptible the measure, the unit in which it would be made was vague
4. **Auto-select tests** - as the user would design its scenario, they desired the tests would automatically be selected if possible to run
5. **Form inputs default values** - some test input fields do not have default values.
6. **Click outside input form field** - there was a bug that if the user clicked outside the input field while filling in the form, the form would close
7. **Feedback while linking elements** - there was no feedback if the user mistakenly linked two elements
8. **Warning for drastic changes** - when clicking in the garbage can, the "New workspace" button, or the home button there was no confirmation to do so
9. **Logic Boxes' conditions** - the values inside the logic boxes would reset every time a user accessed them so would have to input again

10. **Detailed information during test execution** - there is a certain vagueness between the start of the test and the end. The user only sets up the parameters of the test and, in the end, checks if the tests are successful, or not. There is no information regarding sensor readings, or actuator state in-between.

Most of these points were improved in the following days of the usability test. The test detail area (2) was widened so that the messages could be easier perceived and the users wouldn't need to scroll so much as the last iteration. All form input fields (3) were added a '?' icon, like in the test pattern area so that the users can understand the objective of a certain field. Regarding the form inputs, in some were missing the unit in which they would measure, which was also added. Regarding the selection of tests (4), now the user doesn't need to click the checkboxes, present in the test pattern area. As tests become available, they are auto-selected. Only if the user wishes to unselect a test, it clicks in the checkbox regarding that test. Also, a great percentage of users complained about a small aspect regarding the forms, which was upon clicking on the surrounding area of the form it would automatically close it (6), which caused certain displeasure, and it was corrected. Greater feedback when the user attempts to connect blocks was added (7), so the user perceives it is moving in the right direction. Upon clicking in buttons that cause a significant change in the current workspace (8), "Garbage can", "New Workspace" or the logo, the user is present with a confirmation box in order to proceed with that action. The setting up of condition (9) to trigger both action and alerts was criticised due to the fact that the user sometimes, didn't know if he/she had to fill in those fields, but also it was "hidden" and the users felt a bit lost. Now, when the user links a sensor with a logic box, a modal is shown so that the user can, immediately, set the conditions and save in the logic box's parameters.

The complaints regarding the lack of detail, or non-user-friendly error messages and the detailed information during test execution were considered aspects that are out of the domain of this dissertation. Although, being out of the scope, and after a code analysis, it was realized that a significant code refactoring would be necessary in order to implement such features and the time that it would take would put the current project at stake in terms of keeping it within schedule. Therefore, these two features are to be considered as future improvement to the dissertation. The criticism towards the non-existing form default values (5) was decided to not be improved since the tool allows the test of very distinct scenarios, that may have very distinct characteristics. By implementing default values, in a different scenario, the feedback from users could, for example, be to remove such feature, since it would only disturb that scenario's experience.

### 4.2.2 Threats to Validity

The main threats to validation are the following:

- Atmospheric Conditions
- Sensor's and Actuators build quality
- Disparity of values users could choose

- Ability to use Izinto's features (software upgrades)

During the first week of June, which was when the usability test took place, there was a significant change in the temperature and humidity conditions. Since the usability test involved the use of a sensor that performed such readings, there were some tests that didn't end up as planned and instead of obtaining successes, or failures, would take the opposite outcome. Another important factor, was the poor overall quality of the used sensors. During the execution of the usability test, there were some fields that the users could choose out of their own imagination. Since the knowledge of the tool and the overall scenario was relatively small, such inputs could change the success of a test. For example, the choosing of the delay for the action of turning the air conditioning on, if very small (less than 5000 milliseconds) there was a high probability of the test to fail. Lastly, it was not possible to cover all of the available test patterns Izinto had implemented, mainly due to one aspect - the sensor that performs user triggered readings, Nokia weight scale, had a recent software upgrade, from OAuth 1.0 to 2.0, which made all the developed code in Izinto useless. Although it was developed a module that could obtain data from its API, it was not possible to make it work.

# Chapter 5

# Conclusion and Future Work

## 5.1 Conclusion and Contributions

During the development of the dissertation solution it was perceived that, in fact, there are not many solutions for IoT testing, especially with a focus on visual interfaces. There are a few solutions, but their domain is quite limited, or different from the one of this dissertation and very much focused on large-scale systems that don't allow for non-programmers to use due to both their scale and technical knowledge to test their systems.

An overview of the current tools for IoT was made, with focus on the of visual interfaces and it was concluded that the current solutions aren't yet properly addressing the problem it is trying to be solved. There are certain solutions that can, in fact, answer a lot of questions but in a very large-scale systems and using simulated scenarios.

In order to fill in the existing gap in IoT solutions for people with lower technical skill, it was developed a visual interface for IoT testing. Such interface took advantage of an already developed pattern-based testing framework developed on a previous thesis. This interface allows the user to simulate a real scenario, with a set of devices and applications, and perform integration tests with the help of Izinto as an integration testing framework, that runs in the backend. The visual interface also allows for the visualisation of test results.

Finally, the solution was validated with the execution of a usability test, including users with distinct levels of technical skill. The usability test proved that the solution developed was very good, having very good feedback from the users regarding both the easiness of testing an IoT scenario and not needing a high knowledge to use it.

The main contributions of this dissertation are:

- A review of the current state-of-the-art and of the current challenges in the development and testing regarding IoT
- A low-code visual solution which has the objective of reducing both the time required for a user to test an IoT scenario and also to shorten the gap between people with low technical knowledge and knowledgeable users in terms of testing

All of the presented work and referred contributions were summarized in a scientific publication to be submitted in the future.

## 5.2 Future Work

As future work, there are certain factors it is possible to point out, mainly regarding the addition of functionalities to the current solution. There are two paths to follow, one with more focus on the addition of functionalities in Izinto and another one by adding more functionalities to the visual interface and better user experience.

### 5.2.1 Izinto

In terms of addition of features to Izinto, it is possible to identify a set of new patterns to be added. As of now, Izinto covers the test of features. There are more patterns that can, for example, cover the connectivity, performance, scalability of IoT systems, which are usually common among such systems. By covering a greater set of patterns, it is possible to ensure better functioning of such distinct and heterogeneous systems and ensure their integration. There is also the possibility of creating a new set of test patterns for the scope of IoT.

### 5.2.2 Visual Interface

In terms of the visual interface for testing, there is the possibility of making it even more intuitive, and ultimately reducing the time for the design of scenarios. Also, and as mentioned in chapter 4, there is the possibility of creating a module for displaying the sensor readings, or the actuator's state in real time. By doing this, the tester, or user, would feel a more controlled scenario of test and feel in more contact with the actual values being used for test purpose.

Lastly, it could be implemented a system of test management, where past tests could be stored and the users could access them, check their past results and even rerun such tests to see if, by hypothesis, a new test could obtain better results. All this centralised in a database of users and tests and accessible by platform users.

# References


[ABF+16] Abbas Ahmad, Fabrice Bouquet, Elizabeta Fourneret, Franck Le Gall, and Bruno Legeard. Model-based testing as a service for iot platforms. In *International Symposium on Leveraging Applications of Formal Methods*, pages 727–742. Springer, 2016.

[Ato] Atomiton. Atomiton. http://www.atomiton.com/. Accessed: 2019-02-02.

[BK08] Eckard Bringmann and Andreas Krämer. Model-based testing of automotive systems. In *2008 1st international conference on software testing, verification, and validation*, pages 485–493. IEEE, 2008.

[CPN14] Pedro Costa, Ana CR Paiva, and Miguel Nabuco. Pattern based gui testing for mobile applications. In *2014 9th International Conference on the Quality of Information and Communications Technology*, pages 66–74. IEEE, 2014.

[DCPF18] João Pedro Dias, Flávio Couto, Ana CR Paiva, and Hugo Sereno Ferreira. A brief overview of existing tools for testing the internet-of-things. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 104–109. IEEE, 2018.

[Dom] Domatica. easyedge. https://www.easyedge.io/. Accessed: 2019-01-20.

[Fac] FIT Future Internet Testing Facility. Iot-lab. https://www.iot-lab.info/. Accessed: 2019-01-21.

[FFC+18] Bahar Farahani, Farshad Firouzi, Victor Chang, Mustafa Badaroglu, Nicholas Constant, and Kunal Mankodiya. Towards fog-driven iot ehealth: Promises and challenges of iot in medicine and healthcare. *Future Generation Computer Systems*, 78:659–676, 2018.

[Fou] JS Foundation. Node-red. https://nodered.org/. Accessed: 2019-01-23.

[FP14] Felipe Fernandez and George C Pallis. Opportunities and challenges of the internet of things for healthcare: Systems engineering perspective. In *Wireless Mobile Communication and Healthcare (Mobihealth), 2014 EAI 4th International Conference on*, pages 263–266. IEEE, 2014.

[Gmb] Ternary GmbH. Iotify. https://iotify.io/. Accessed: 2019-01-20.

[GMPE13] Pablo Giménez, Benjamin Molína, Carlos E Palau, and Manuel Esteve. Swe simulation and testing for the iot. In *Systems, man, and cybernetics (SMC), 2013 IEEE international conference on*, pages 356–361. IEEE, 2013.

# REFERENCES


[GVDGB17] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296, 2017.

[HLC+14] Son N Han, Gyu Myoung Lee, Noel Crespi, Kyongwoo Heo, Nguyen Van Luong, Mihaela Brut, and Patrick Gatellier. Dpwsim: A simulation toolkit for iot applications using devices profile for web services. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 544–547. IEEE, 2014.

[Joi] JointJS. Jointjs. https://www.jointjs.com/. Accessed: 2019-05-19.

[KH16] X Krasniqi and E Hajrizi. Use of iot technology to drive the automotive industry from connected to full autonomous vehicles. *IFAC-PapersOnLine*, 49(29):269–274, 2016.

[LF16] Bruno Lima and João Pascoal Faria. A survey on testing distributed and heterogeneous systems: The state of the practice. In *International Conference on Software Technologies*, pages 88–107. Springer, 2016.

[LF17] Bruno Miguel Carvalhido Lima and João Carlos Pascoal Faria. Towards decentralized conformance checking in model-based testing of distributed systems. In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 356–365. IEEE, 2017.

[LLTT12] Yu Beng Leau, Wooi Khong Loo, Wai Yip Tham, and Soo Fun Tan. Software development life cycle agile vs traditional approaches. In *International Conference on Information and Network Technology*, volume 37, pages 162–167, 2012.

[LLWC03] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 126–137. ACM, 2003.

[LODYJ12] Vilen Looga, Zhonghong Ou, Yang Deng, and Antti Yla-Jaaski. Mammoth: A massive-scale emulation platform for internet of things. In *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on*, volume 3, pages 1235–1239. IEEE, 2012.

[MK16] Nitinder Mohan and Jussi Kangasharju. Edge-fog cloud: A distributed cloud for internet of things computations. In *2016 Cloudification of the Internet of Things (CIoT)*, pages 1–6. IEEE, 2016.

[MQT] MQTT. Mqtt. http://mqtt.org/. Accessed: 2019-06-09.

[MSB11] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[Noda] Node.js. Node.js. https://nodejs.org/en/. Accessed: 2019-05-19.

[Nodb] NodeMailer. Nodemailer. https://nodemailer.com/about/. Accessed: 2019-01-23.

# REFERENCES


[NPM] NPM. Node package manager. https://www.npmjs.com/. Accessed: 2019-01-23.

[PKSL16] Tamas Pflanzner, Attila Kertész, Bart Spinnewyn, and Steven Latré. Mobiotsim: towards a mobile iot device simulator. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pages 21–27. IEEE, 2016.

[PLF18] Pedro Martins Pontes, Bruno Lima, and João Pascoal Faria. Izinto: a pattern-based iot testing framework. In *Companion Proceedings for the ISSTA/ECOOP 2018 Workshops*, pages 125–131. ACM, 2018.

[Plu] PIO Plus. Platformio. http://platformio.org/. Accessed: 2019-01-20.

[PRS02] J. Preece, Y. Rogers, and H. Sharp. *Interaction design: beyond human-computer interaction*. J. Wiley & Sons, 2002.

[RWBO15] Philipp Rosenkranz, Matthias Wählisch, Emmanuel Baccelli, and Ludwig Ortmann. A distributed test system architecture for open-source iot software. In *Proceedings of the 2015 Workshop on IoT challenges in Mobile and Industrial Systems*, pages 43–48. ACM, 2015.

[Sim] SimpleSoft. Simpleiotsimulator. https://www.smplsft.com/SimpleIoTSimulator.html. Accessed: 2019-01-23.

[SLS14] Andreas Spillner, Tilo Linz, and Hans Schaefer. *Software testing foundations: a study guide for the certified tester exam*. Rocky Nook, Inc., 2014.

[Sof] Indium Software. Iot automated testing. https://www.indiumsoftware.com/blog/iot-automated-testing/. Accessed: 2019-06-04.

[Spo] Alexander Spotnitz. Moving the cloud to the edge. https://www.pubnub.com/blog/moving-the-cloud-to-the-edge-computing/. Accessed: 2019-06-20.

[TM17] Antero Taivalsaari and Tommi Mikkonen. A roadmap to the programmable world: software challenges in the iot era. *IEEE Software*, 34(1):72–80, 2017.

[Wel38] Bernard L Welch. The significance of the difference between two means when the population variances are unequal. *Biometrika*, 29(3/4):350–362, 1938.

# REFERENCES

# Appendix A

# Usability Test

## A.1 Usability Test Handout

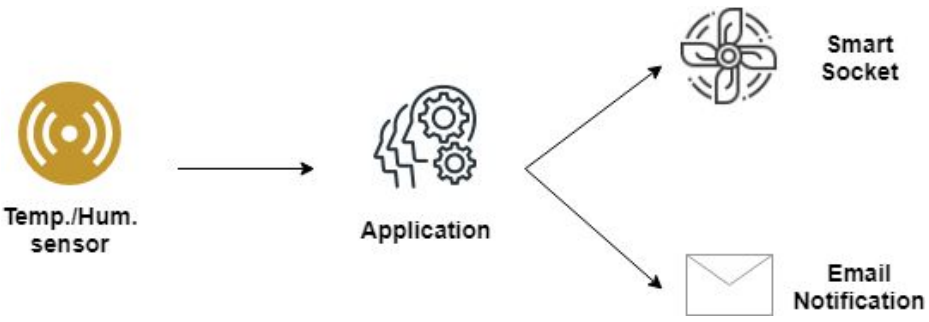
For the usability test it was handed to the users a handout, as it it possible to observe in the next pages. The user would follow it in order to complete the tasks. It provides some useful information regarding the way to execute them.

# Domotics Scenario Usability Test

First of all, **thank you** for your participation in this usability test.

In the following section, you will be asked to execute some short tasks in order to assess the utility and quality of the work developed on a master's thesis.
The objective of this tool is to lower both the technical knowledge necessary and the time spent in order to testing an IoT scenario.
Remember, you may end the test at any time.
Feel free to ask any questions.

## Current Scenario




The current scenario includes a sensor capable of reading both temperature and humidity. The sensor is connected to an application capable of turning on and off a smart socket and also notify the user by sending an email.
It is also possible to test, individually, if the sensor and the smart socket (actuator) are working as intended.

Watch a short video to understand the general concepts of the application.

Start the form: https://forms.gle/H8PBUjco6saRGbZFA before initiating the test.

# Tasks

**Task 1:** Import an already set up scenario, run the available test and interpret results.

- **Context:** You want to run, once again, an already designed scenario to check if, hypothetically, you can obtain better results than the past ones.

Complete the task and speak your thoughts aloud.

**Steps:**

1. On the right-top side of the interface, there are a group of 4 buttons. Click on the one with an upper arrow icon 
2. Locate, on the computer, the file with the scenario configuration - **periodic.json**
3. After importing, click on the check box referring to the Periodic Reading test on the test pattern area 

4. In the area of the 4 buttons, click on the green one, with the icon “Play” 
5. Wait for the test to finish and verify which “Checks” passed or failed.

- Please answer the form regarding this task.

**Task 2:** Test the correct functioning of the smart socket.

- **Context:** You want to test the correct functioning of the smart socket. You will do that by connecting a lamp to the socket and testing a series of state changing events (ON -> OFF, and vice-versa).

Complete the task and speak your thoughts aloud.

**Steps:**

1. Create a new Workspace in the green button on the top-center of the screen.
2. Go to the Tool Box (left side) and click the Actuator button

| Field | (Interval of) Values |
|---|---|
| Actuator name | sonoff |
| Actuator Communication | MQTT Actuator |
| Purpose of Actuator | Test the Actuator behavior |
| Number of state changes | 1+ |
| MQTT Broker Address | tcp://10.227.107.152 |
| MQTT State Topic | stat/sonoff/POWER |
| Acceptable Delay | 2000+ |
| State changing command | cmnd/sonoff/power |
| Argument to turn ON | 1 |
| Argument to turn OFF | 0 |

3. Run the test available and observe the results

- Please answer the form regarding this task.

**Task 3:** Test the correct functioning of the temperature and humidity sensor.

- **Context:** You want to test the temperature and humidity to check that it delivers periodic readings within a certain interval of time.

Complete the task and speak your thoughts aloud.

**Steps:**

1. Create a new Workspace in the green button on the top-center of the screen.
2. In the sensor creation form, fill in the fields as follows

| Field | (Interval of) Values |
|---|---|
| Sensor Name | am2302_01 |
| Types of readings | Temperature, Humidity |
| Expected Interval of Time (milliseconds) | 60 000 |
| Acceptable Deviation (milliseconds) | 3000+ |
| Acceptable Delay | 2000+ |
| Temperature Value Specification - Minimum value (ºC) | [10-20] |
| Temperature Value Specification - Maximum value (ºC) | [21-40] |
| Temperature Value Specification - Reading Trend | No Trend |
| Humidity Value Specification - Minimum value (%) (0-100) | [0-40] |
| Humidity Value Specification - Maximum value (%) (0-100) | [41-100] |
| Humidity Value Specification - Reading Trend | No Trend |

3. Run the test and observe results.

- Please answer the form regarding this task.

**Task 4:** Test the **action** of toggling ON of an air conditioning in the event of temperature increase in your workplace.

- **Context:** You want to test an Android application, a smart socket and a temperature sensor, expecting them to trigger the toggling on of an air conditioning when the temperature of the room reaches a certain value.

Complete the task and speak your thoughts aloud.

**Steps:**

1. Create a new Workspace in the green button on the top-center of the screen.
2. Create a sensor that reads temperature, named “am2302_01” and complete all values of the form

| Field | (Interval of) Values |
|---|---|
| Sensor Name | am2302_01 |
| Types of readings | Temperature |
| Expected Interval of Time (milliseconds) | 60 000 |
| Acceptable Deviation (milliseconds) | 3000+ |
| Acceptable Delay | 5000+ |
| Temperature Value Specification - Minimum value (ºC) | [10-20] |
| Temperature Value Specification - Maximum value (ºC) | [25-40] |
| Temperature Value Specification - Reading Trend | Random |

3. Create an Logic Box with following parameters

| Field | (Interval of) Values |
|---|---|
| Name | Any |
| Types of outcome | Action |
| Acceptable Delay | 20 000+ |
| Description | (TRIGGERED) Temperature higher than 30 |
| Expected Initial State (Socket) | OFF |
| Expected Final State (Socket) | ON |

| | |
|---|---|
| Trigger Specification | Any |

4. Link the **Sensor** to the **Logic Box** (click in the black ball and drag to the logic box)

Example: 


5. Double click the Logic Box and set up the conditions for the triggering of the action - new section of the form appeared in the bottom.
6. Create an actuator with following values

| Field | (Interval of) Values |
|---|---|
| Actuator name | sonoff |
| Actuator Communication | MQTT Actuator |
| Purpose of Actuator | Abstraction for outcome |
| MQTT Broker Address | tcp://10.227.107.152 |
| MQTT State Topic | stat/sonoff/POWER |
| Time | 0 |
| Message | Any |
| Command | cmnd/sonoff/power |
| Arguments | 0 |

7. Link **Logic Box** with **Actuator**, as shown before
8. Start action test execution.
9. Use some creative and safe way to increase the temperature read by the sensor
10. Observe test results

- Please answer the form regarding this task.

**Task 5:** Test an **alert** by sending of an email when the event of high temperature and low humidity in your workplace.

- **Context:** You want to test an Android application, an email notification system and a temperature/humidity sensor, expecting them to send an email when the temperature and humidity of your workplace reach certain values.

- This task has the same structure of the last task, only it does **not** include an actuator, but an email block.

| Field | (Interval of) Values |
|---|---|
| Description | (TRIGGERED) Temperature higher than 30 |
| Email | healthkaa@gmail.com |
| Password | ********** |

Complete the task and speak your thoughts aloud.

- Please answer the form regarding this task.

## A.2 Usability Test Time Logging

In order to record the time taken by each user and to analyse significant difference in task completion, a timestamp sheet would be filled in, by a facilitator, as the user completed the task. The sheet is present in the next pages.

## Timestamping:

**Task 1:** Import an already set up scenario, run the available test and interpret results.

| **Sub-Tasks** | **Person11** | **Person12** | **Person13** | **Person14** | **Person15** |
|---|---|---|---|---|---|
| Click on the upload button | | | | | |
| Select JSON file from computer | | | | | |
| Check Periodic Reading test | | | | | |
| **Total amount time:** | | | | | |
| **Average:** | | | | | |
| **Observations:** | | | | | |

**Task 2:** Test the correct functioning of the smart socket.

| **Sub-Tasks** | **Person11** | **Person12** | **Person13** | **Person14** | **Person15** |
|---|---|---|---|---|---|
| Select the “Actuator” from the devices’ menu | | | | | |
| Completely fills in the actuator form with values desired | | | | | |
| Selects the appropriate test to run | | | | | |
| **Total amount time:** | | | | | |
| **Average:** | | | | | |
| **Observations:** | | | | | |

| |
|---|
| |

**Task 3:** Test the correct functioning of the temperature and humidity sensor.

| **Sub-Tasks** | **Person11** | **Person12** | **Person13** | **Person14** | **Person15** |
|---|---|---|---|---|---|
| Select the “Sensor” from the devices’ menu | | | | | |
| Completely fills in the sensor form with values desired and choosing Temperature and Humidity as sensor readings | | | | | |
| Selects the appropriate test to run | | | | | |
| **Total amount time:** | | | | | |
| **Average:** | | | | | |
| **Observations:** | | | | | |

**Task 4:** Test the toggling ON of an air conditioning in the event of high temperature in your workplace.

| **Sub-Tasks** | **Person11** | **Person12** | **Person13** | **Person14** | **Person15** |
|---|---|---|---|---|---|
| Selects the “Sensor” from the devices’ menu | | | | | |

| | | | | | |
|---|---|---|---|---|---|
| Completely fills in the sensor form with values desired, choosing “Temperature” as readings | | | | | |
| Selects the “App Box” from the devices’ menu | | | | | |
| Completely fills in the app form with values desired | | | | | |
| Links the sensor with App Box | | | | | |
| Enters in the edit form of the App Box and chooses the conditions. | | | | | |
| Selects the “Actuator” from the devices’ menu | | | | | |
| Completely fills in the actuator form with values desired | | | | | |
| Links the App Box with the actuator | | | | | |
| Selects the appropriate test to run | | | | | |
| **Total amount time:** | | | | | |
| **Average:** | | | | | |
| **Observations:** | | | | | |

**Task 5:** Test the sending of an email when the event of high temperature and low humidity in your workplace.

| **Sub-Tasks** | **Person11** | **Person12** | **Person13** | **Person14** | **Person15** |
|---|---|---|---|---|---|
| Selects the “Sensor” from the devices’ menu | | | | | |

| | | | | | |
|---|---|---|---|---|---|
| Completely fills in the sensor form with values desired, choosing Temperature and Humidity as sensor readings | | | | | |
| Selects the “App Box” from the devices’ menu | | | | | |
| Completely fills in the app form with values desired | | | | | |
| Links the sensor with App Box | | | | | |
| Enters in the edit form of the App Box and chooses the conditions for temperature and humidity | | | | | |
| Selects the “Email” from the devices’ menu | | | | | |
| Completely fills in the email form with values desired | | | | | |
| Links the App Box with the email block | | | | | |
| Selects the appropriate test to run | | | | | |
| **Total amount time:** | | | | | |
| **Average:** | | | | | |
| **Observations:** | | | | | |

## A.3 Usability Test User Form

With the objective of perceiving the users' feedback and register the data collected regarding users feedback, a form would be present to the user. In the next pages, there is a copy of the form.

# Usability Test - Hugo Cunha MSc

This form has the purpose of understanding the level of utility, easiness and simplicity of the solution implemented during the MSc of Hugo Cunha.
Please execute the task in the first place and then answer the questionnaire regarding that task.
In the end, answer a final questionnaire regarding the whole experience.
Move to the next section for a short initial questionnaire.

*Obrigatório

## Initial Questionnaire

1. **How would you evaluate your technical knowledge in programming** *
   *Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Low to none | ◯ | ◯ | ◯ | ◯ | ◯ | Very knowledgeable |

2. **For how long have you been programming (years)** *

_____

3. **What is your knowledge regarding testing software-based systems?** *
   *Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Low to none | ◯ | ◯ | ◯ | ◯ | ◯ | Very knowledgeable |

4. **What is your knowledge regarding IoT (Internet-of-Things)?** *
   *Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Low to none | ◯ | ◯ | ◯ | ◯ | ◯ | Very knowledgeable |

## Please execute Task 1 before moving to the next section of the form

## Task 1 Form

5. **How easy it was, in general, to execute the task?** *
   *Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Very difficult | ◯ | ◯ | ◯ | ◯ | ◯ | Very easy |

6. **How pleasant it was, in general, to execute the task?** *
*Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Very unpleasant | ◯ | ◯ | ◯ | ◯ | ◯ | Very pleasant |

7. **Please point out any difficulty you may have found during the execution of the task or indicate any suggestion to improve the experience.** *

## Please execute Task 2 before moving to the next section of the form

## Task 2 Form

8. **How easy it was, in general, to execute the task?** *
*Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Very difficult | ◯ | ◯ | ◯ | ◯ | ◯ | Very easy |

9. **How pleasant it was, in general, to execute the task?** *
*Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Very unpleasant | ◯ | ◯ | ◯ | ◯ | ◯ | Very pleasant |

10. **Please point out any difficulty you may have found during the execution of the task or indicate any suggestion to improve the experience.** *

## Please execute Task 3 before moving to the next section of the form

## Task 3 Form

11. **How easy it was, in general, to execute the task?** *
*Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Very difficult | ◯ | ◯ | ◯ | ◯ | ◯ | Very easy |

12. **How pleasant it was, in general, to execute the task?** *
*Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Very unpleasant | ◯ | ◯ | ◯ | ◯ | ◯ | Very pleasant |

13. **Please point out any difficulty you may have found during the execution of the task or indicate any suggestion to improve the experience.** *

## Please execute Task 4 before moving to the next section of the form

## Task 4 Form

14. **How easy it was, in general, to execute the task?** *
*Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Very difficult | ◯ | ◯ | ◯ | ◯ | ◯ | Very easy |

15. **How pleasant it was, in general, to execute the task?** *
*Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Very unpleasant | ◯ | ◯ | ◯ | ◯ | ◯ | Very pleasant |

16. **Please point out any difficulty you may have found during the execution of the task or indicate any suggestion to improve the experience.** *

# Please execute Task 5 before moving to the next section of the form

## Task 5 Form

17. **How easy it was, in general, to execute the task?** *
*Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Very difficult | ◯ | ◯ | ◯ | ◯ | ◯ | Very easy |

18. **How pleasant it was, in general, to execute the task?** *
*Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Very unpleasant | ◯ | ◯ | ◯ | ◯ | ◯ | Very pleasant |

19. **Please point out any difficulty you may have found during the execution of the task or indicate any suggestion to improve the experience.** *

## Final Questionnaire

20. **Did you find it easy to test IoT systems with this tool? Do you think this is an useful tool?** *
*Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly aggree |

21. **In comparison to manual coding of tests, did you find it easier to create tests using this tool?** *
*Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly aggree |

22. **How easy it was to run the tests and interpret the test results?** *
*Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Very difficult | ◯ | ◯ | ◯ | ◯ | ◯ | Very easy |

23. **Do you think this tool could be used by people with low or none programming experience?** *

*Marcar apenas uma oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly aggree |

24. **Is there something you would suggest in order to improve the application?** *